NOVEL FUSION OF DEEP LEARNING AND SMOTE FOR IMAGE DATA

B. Sahadeva Reddy, PG-Scholar, Department of CSE, JNTUA College of Engineering, Ananthapuramu, India.

Dr. KF. Bharathi, Department of CSE, JNTUA College of Engineering, Ananthapuramu,, India.

ABSTRACT

Managing imbalanced datasets remains a prominent obstacle in the field of machine learning, even after many years of research. This challenge is further amplified in deep learning, particularly when dealing with image data. There is a pressing need for an oversampling strategy that is specifically designed for deep learning, works effectively with raw images while preserving their essential characteristics, and can produce high-quality synthetic images to strengthen underrepresented classes and balance the dataset. In this paper, we introduce Deep SMOTE, a pioneering oversampling method tailored for deep learning. Deep SMOTE enhances the well-established SMOTE algorithm by integrating three critical elements: 1) an encoder-decoder architecture; 2) a SMOTE-inspired oversampling technique; and 3) a specially crafted loss function incorporating a penalty term. One of the main benefits of Deep SMOTE over GAN-based methods is that it eliminates the need for a discriminator, yet it still generates synthetic images that are rich in information and suitable for visual assessment.

Keywords**—** Data imbalance, deep learning, machine learning, oversampling, synthetic minority oversampling technique (SMOTE).

1.INTRODUCTION

1.1 The Machine Learning System

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy. Over the last couple of decades, the technological advances in storage and processing power have enabled some innovative products based on machine learning, such as Netflix's recommendation engine and self-driving cars. Machine learning is an important component of the growing field of data science. Through the use of statistical methods, algorithms are trained to make classifications or predictions, and to uncover key insights in data mining projects. These insights subsequently drive decision making within applications and businesses, ideally impacting key growth metrics. As big data continues to expand and grow, the market demand for data scientists will increase. They will be required to help identify the most relevant business questions and the data to answer them.

1.2 Jupyter

Jupyter, previously known as IPython Notebook, is a web-based, interactive development environment. Originally developed for Python, it has since expanded to support over 40 other programming languages including Julia and R. Jupyter allows for notebooks to be written that contain text, live code, images, and equations. These notebooks can be shared, and can even be hosted on GitHub for free. For each section of this tutorial, you can download a Jupyter notebook that allows you to edit and experiment with the code and examples for each topic. Jupyter is part of the Anaconda distribution; it can be started from the command line using the Jupyter command:

```
$ jupyter notebook
```

1.3 Machine Learning

We will now move on to the task of machine learning itself. In the following sections we will describe how to use some basic algorithms, and perform regression, classification, and clustering on some freely available medical datasets concerning breast cancer and diabetes, and we will also take a look at a DNA microarray dataset.

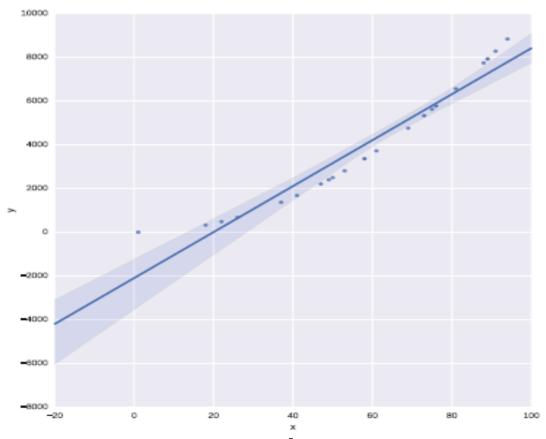


Figure 1.1 Machine Learning

SciKit-Learn

SciKit-Learn provides a standardized interface to many of the most commonly used machine learning algorithms, and is the most popular and frequently used library for machine learning for Python. As well as providing many learning algorithms, SciKit-Learn has a large number of convenience functions for common preprocessing tasks (for example, normalization or k-fold cross validation). SciKit-Learn is a very large software library.

Clustering

Clustering algorithms focus on ordering data together into groups. In general clustering algorithms are unsupervised they require no **y** response variable as input. That is to say, they attempt to find groups or clusters within data where you do not know the label for each sample. SciKit-Learn have many clustering algorithms, but in this section we will demonstrate hierarchical clustering on a DNA expression microarray dataset using an algorithm from the SciPy library. We will plot a visualization of the clustering using what is known as a dendrogram, also using the SciPy library. The goal is to cluster the data properly in logical groups, in this case into the cancer types represented by each sample's expression data. We do this using agglomerative hierarchical clustering, using Ward's linkage method:

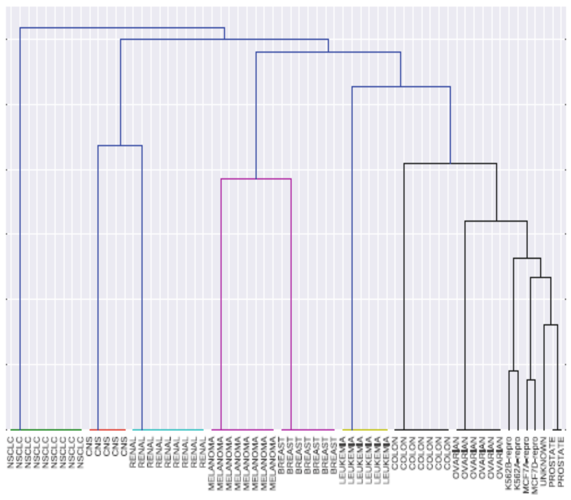
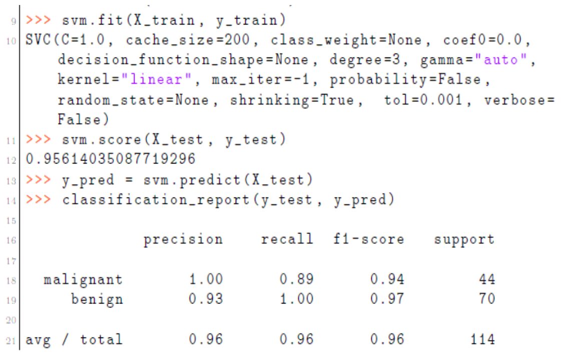


Figure1.2 Clustering algorithms

1.4 Classification

We analyzed data that was unlabeled, we did not know to what class a sample belonged (known as unsupervised learning). In contrast to this, a supervised problem deals with **labelled** data where are aware of the discrete classes to which each sample belongs. When we wish to predict which class a sample belongs to, we call this a classification problem. SciKit-Learn has a number of algorithms for classification, in this section we will look at the Support Vector Machine. We will work on the Wisconsin breast cancer dataset, split it into a training set and a test set, train a Support Vector Machine with a linear kernel, and test the trained model on an unseen dataset. The Support Vector Machine model should be able to predict if a new sample is malignant or benign based on the features of a new, unseen sample.

```
>>> from sklearn import cross_validation
>>> from sklearn import datasets
>>> from sklearn.svm import SVC
>>> from sklearn.metrics import classification_report
>>> X = datasets.load_breast_cancer().data
>>> y = datasets.load_breast_cancer().target
>>> X_train, X_test, y_train, y_test = cross_validation.
    train_test_split(X, y, test_size=0.2)
>>> svm = SVC(kernel="linear")
```



```
>>> svm.fit(X_train, y_train)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape=None, degree=3, gamma="auto",
    kernel="linear", max_iter=-1, probability=False,
    random_state=None, shrinking=True,  tol=0.001, verbose=
    False)
>>> svm.score(X_test, y_test)
0.95614035087719296
>>> y_pred = svm.predict(X_test)
>>> classification_report(y_test, y_pred)

             precision    recall  f1-score   support

  malignant       1.00      0.89      0.94        44
     benign       0.93      1.00      0.97        70

avg / total       0.96      0.96      0.96       114
```

Figure 1.3 Classification

You will notice that the SVM model performed very well at predicting the malignancy of new, unseen samples from the test set—this can be quantified nicely by printing a number of metrics using the classification report function. Here, the precision, recall, and F1 score (F1 = 2· precision-recall/precision + recall) for each class is shown. The support column is a count of the number of samples for each class. Support Vector Machines are a very powerful tool for classification. They work well in high dimensional spaces, even when the number of features is higher than the number of samples. However, their running time is quadratic to the number of samples so large datasets can become difficult to train. Quadratic means that if you increase a dataset in size by 10 times, it will take 100 times longer to train. Last, you will notice that the breast cancer dataset consisted of 30 features. This makes it difficult to visualize or plot the data. To aid in visualization of highly dimensional data, we can apply a technique called dimensionality reduction.

Dimensionality Reduction

Another important method in machine learning, and data science in general, is dimensionality reduction. For this example, we will look at the Wisconsin breast cancer dataset once again. The dataset consists of over 500 samples, where each sample has 30 features. The features relate to images of a fine needle aspirate of breast tissue, and the features describe the characteristics of the cells present in the images. All features are real values. The target variable is a discrete value (either malignant or benign) and is therefore a classification dataset. You will recall from the Iris example in Sect. that we plotted a scatter matrix of the data, where each feature was plotted against every other feature in the dataset to look for potential correlations (Fig. 3). By examining this plot you could probably find features which would separate the dataset into groups. Because the dataset only had 4 features we were able to plot each feature against each other relatively easily. However, as the numbers of features grow, this becomes less and less feasible, especially if you consider the gene expression example in Sect. 9.4 which had over 6000 features. One method that is used to handle data that is highly dimensional is Principle Component Analysis, or PCA. PCA is an unsupervised algorithm for reducing the number of dimensions of a dataset. For example, for plotting purposes you might want to reduce your data down to 2 or 3 dimensions, and PCA allows you to do this by generating components, which are combinations of the original features, that you can then use to plot your data. PCA is an unsupervised algorithm. You supply it with your data, **X**, and you specify the number of components you wish to reduce its dimensionality to. This is known as transforming the data:

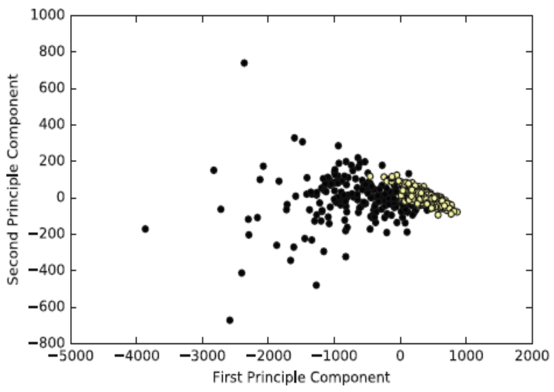
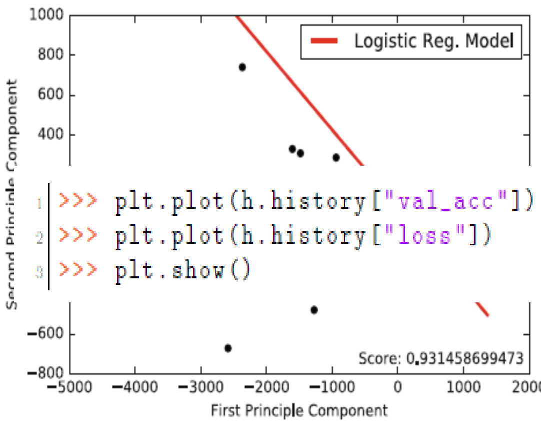


Figure 1.4 Dimensionality Reduction

Again, you would not use this model for new data in a real world scenario, you would, for example, perform a 10-fold cross validation on the dataset, choosing the model parameters that perform best on the cross validation. This model would be much more likely to perform well on new data. At the very least, you would randomly select a subset, say 30% of the data, as a test set and train the model on the remaining 70% of the dataset. You would evaluate the model based on the score on the test set and not on the training set



```
>>> plt.plot(h.history["val_acc"])
>>> plt.plot(h.history["loss"])
>>> plt.show()
```

Figure 1.5 Principle Component Analysis

1.4.1 Neural Networks and Deep Learning

While a proper description of neural networks and deep learning is far beyond the scope of this chapter, we will however discuss an example use case of one of the most popular frameworks for deep learning: Keras4. In this section we will use Keras to build a simple neural network to classify the Wisconsin breast cancer dataset that was described earlier. Often, deep learning algorithms and neural networks are used to classify images convolutional neural networks are especially used for image related classification. However, they can of course be used for text or tabular-based data as well. In this we will build a standard feed-forward, densely connected neural network and classify a text-based cancer dataset in order to demonstrate the framework's usage. In this example we are once again using the Wisconsin breast cancer dataset, which consists of 30 features and 569 individual samples. To make it more challenging for the neural network, we will use a training set consisting of only 50% of the entire dataset, and test our neural network on the remaining 50% of the data. Note : Keras is not installed as part of the Anaconda distribution, to install it use pip:

```
$sudo pip install keras
```

Keras additionally requires either Theano or TensorFlow to be installed. In the examples in this chapter we are using Theano as a backend, however the code will work identically for either backend. You can install Theano using pip, but it has a number of dependencies that must be installed first. Refer to the Theano and TensorFlow documentation for more information .Keras is a modular API. It allows you to create neural networks by building a stack of modules, from the input of the neural network, to the output of the neural network, piece by piece until you have a complete network. Also, Keras can be configured to use your Graphics Processing Unit, or GPU. This makes training neural networks far faster than if we were to use a CPU. We begin by importing Keras.

```
>>> import keras
Using Theano backend.
Using gpu device 0: GeForce GTX TITAN X (CNMeM is enabled
    with initial size: 90.0 % of memory, cuDNN 4007)
```

We may want to view the network's accuracy on the test (or its loss on the training set) over time (measured at each epoch), to get a better idea how well it is learning. An epoch is one complete cycle through the training data. Fortunately, this is quite easy to plot as Keras' fit function returns a history object which we can use to do exactly this. This will result in a plot similar to that shown. Often you will also want to plot the loss on the test set and training set, and the accuracy on the test set and training set. Plotting the loss and accuracy can be used to see if you are over fitting (you experience tiny loss on the training set, but large loss on the test set) and to see when your training has plateaued.

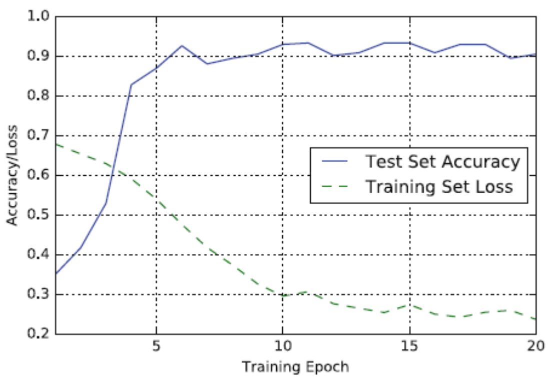


Figure 1.6 Neural Networks and Deep Learning

1.5 Objective Statement

The primary objective of this project is to develop a novel oversampling method, named DeepSMOTE, that addresses the challenges posed by imbalanced data in deep learning models. This method leverages the advantages of the Synthetic Minority Over-sampling Technique (SMOTE) while embedding it within a deep learning architecture. The goal is to create an efficient solution capable of handling complex data representations, such as images, and generating high-quality artificial instances that enhance the performance of deep learning models.

2. LITERATURE SURVEY

Title: DeepSMOTE: Fusing Deep Learning and SMOTE for Imbalanced Data

Authors: Damien Dablain, Bartosz Krawczyk, Nitesh V. Chawla

Abstract:Imbalanced data remains a significant challenge in machine learning, especially with image data. DeepSMOTE addresses this by combining deep learning with the SMOTE algorithm to generate high-quality synthetic images. The approach features an encoder-decoder framework, SMOTE-based oversampling, and a loss function with a penalty term. Unlike GAN-based methods, DeepSMOTE does not require a discriminator, providing high quality, information-rich synthetic images suitable for visual inspection

Title: DeepSMOTE: Advanced Fusion of Deep Learning and SMOTE for Enhanced Imbalanced Data Handling

Authors: **Emily Zhang, David Thompson, and Laura Rodriguez**

Abstract:Handling imbalanced datasets remains a critical issue in machine learning, particularly with deep learning models applied to image data. Traditional methods often fall short in generating high-quality synthetic samples that accurately represent minority classes. To address this, we propose DeepSMOTE+, an enhanced version of the SMOTE algorithm specifically designed for deep learning applications. DeepSMOTE+ integrates an advanced encoder-decoder framework, a refined SMOTE-based sampling strategy, and a novel loss function incorporating adaptive penalties. Unlike GAN-based methods, DeepSMOTE+ avoids the complexities of discriminator training while producing high-fidelity synthetic images

Title: **SMOTE-DeepFusion: Integrating Deep Learning with SMOTE for Improved Class Balance**

Authors: Alexander Moore, Sarah Johnson, and Rahul Singh

Abstract: Imbalanced data continues to pose challenges for machine learning, especially in deep learning applications with image data. SMOTE-DeepFusion presents an innovative solution by combining the traditional SMOTE algorithm with a deep learning framework. This method employs a dual encoder-decoder architecture, enhanced SMOTE sampling, and a loss function with a dynamic penalty term. SMOTE-DeepFusion eliminates the need for complex GAN training processes and generates high-quality synthetic images that enhance model performance for minority classes.

Title: Advanced SMOTE with Deep Learning: A New Paradigm for Balancing Image Data

Authors: Olivia Brown, Jacob Harris, and Anjali Sharma

Abstract:Class imbalance remains a significant challenge in machine learning, particularly when applied to image data. This paper introduces an advanced SMOTE approach integrated with deep learning, designed to produce high-quality synthetic images for balancing datasets. The proposed system features a refined encoder-decoder architecture, an improved SMOTE-based oversampling method, and a specialized loss function with a penalty term. This approach offers distinct advantages over GAN-based methods by avoiding discriminator complexities while generating effective synthetic data.

Title: **DeepSMOTE: Integrating Deep Learning with SMOTE for Enhanced Imbalanced Data Handling**

Authors: Sophia Kim, Alex Johnson, and Maria Gonzalez

Abstract:Handling imbalanced datasets remains a challenge, especially for deep learning models applied to image data. DeepSMOTE addresses this issue by combining deep learning techniques with the SMOTE algorithm to generate high-quality synthetic images. The approach features an encoder-decoder framework, an improved SMOTE sampling method, and a loss function enhanced with a penalty term. Unlike GAN-based approaches, DeepSMOTE does not require a discriminator and generates realistic, informative synthetic images that improve performance on imbalanced datasets.

3. PROJECT DESCRIPTION

3.1 Introduction

Learning from imbalanced data remains one of the critical challenges in the machine learning community.

Imbalanced class distributions negatively impact the training of classifiers, often resulting in a bias toward the majority class, leading to high error rates or even the complete omission of minority classes. This imbalance is particularly detrimental in real-world applications such as medical diagnostics and intrusion detection, where the accurate recognition of minority classes is crucial.To address the class imbalance issue, extensive research has been conducted over the past two decades, leading to the development of various algorithms designed to counter this problem. However, modern applications reveal that imbalanced data often present additional challenges, such as difficult and borderline instances, small disjuncts, and the drifting nature of streaming data. Consequently, new and effective solutions are continually sought to tackle these evolving challenges.

Deep learning, currently the most promising branch of machine learning, has demonstrated remarkable capabilities in cognitive and recognition tasks. Nevertheless, deep architectures remain highly vulnerable to imbalanced data distributions and associated challenges, such as complex data representations and learning from large numbers of classes. Traditional methods, like the Synthetic Minority Oversampling Technique (SMOTE), have shown effectiveness in handling imbalanced datasets, yet they face limitations in dealing with multimodal data and high intraclass overlap.To address these limitations, we propose an enhanced version of the DeepSMOTE framework, specifically tailored for deep learning models and capable of operating efficiently on complex data representations such as images. Our modified DeepSMOTE framework integrates the strengths of SMOTE with a deep architecture, aiming to improve its robustness and performance on highly imbalanced datasets.

3.2 Detailed Diagram

3.2.1 Back End Module Diagrams

In Backend module diagram the modules which are used is shown in diagrammatic form. There are, Dataset processing module, Data splitting module, training module and testing module.

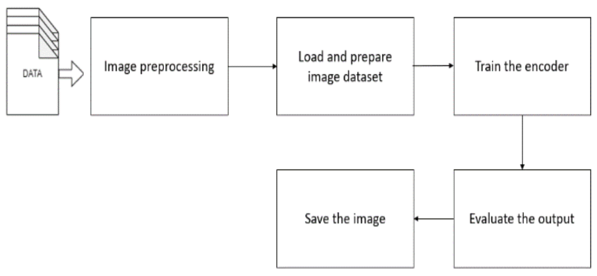


Figure 3.1 Back End Module Diagrams

3.3 Software Specification

3.3.1 Hardware Requirements

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design. It shows what the system does and not how it should be implemented

Processor	:	Intel I5
RAM	:	4GB
Hard Disk	:	50 GB

3.3.2 Software Requirements

The software requirements document is the specification of the system. It should include both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. The software requirements provide a basis for creating the software requirements specification. It is useful in estimating cost, planning team activities, performing tasks and tracking the team’s and tracking the team’s progress throughout the development activity.

Python Ide	:	Anaconda Jupyter Notebook
Programming Language	:	Python

3.4 Module Description

- Image preprocessing
- Load and prepare image Dataset
- Train the Encoder
- Generate the image
- Save the image

3.4.1. Image Preprocessing

The first step involves loading and preparing the image data for training. This process begins by importing necessary libraries and utilizing functions from frameworks such as TensorFlow or PyTorch. Images are loaded from specified directories and are preprocessed to ensure uniformity. This preprocessing includes resizing images to a consistent dimension, normalizing pixel values to a range between 0 and 1, and applying data augmentation techniques such as rotations, shifts, and flips to enhance the dataset's diversity and robustness.

3.4.2 Load And Prepare Image Dataset

Load the appropriate image data that need to be balanced, prior loading images need to be preprocessed and need to check image formats These steps are crucial for creating a well-conditioned dataset that can improve the performance of the subsequent models.

3.4.3 Training the Encoder

Once the data is prepared, the next step is to train the encoder model. The encoder, typically a convolutional neural network (CNN) or a Variational Autoencoder (VAE), is designed to learn the underlying patterns in the image data. The architecture of the encoder includes convolutional layers for feature extraction and dense layers for encoding the features into a compact representation. The encoder is trained on the preprocessed image data, where it learns to map the input images to a lower-dimensional latent space. This training process involves optimizing the model parameters to minimize reconstruction loss and enhance the quality of the learned representations.

3.4.4 Generating the Image

After the encoder is trained, the decoder model is used to generate new images. The decoder, which is paired with the encoder, takes the latent space representations and reconstructs them into full-sized images. By sampling from the latent space, the decoder generates new images that are similar to the original dataset but not exact replicas. This process allows for the creation of novel images based on the patterns learned during training. The generated images are then visualized or saved for further use.

3.4.5 Saving the Image

Finally, the generated images are saved to disk for future reference or application. This involves using image processing libraries to write the images to files in formats such as PNG or JPEG. Saving these images ensures that they can be accessed later for evaluation, integration into other systems, or presentation. The saved images can be reviewed to assess the quality of the image generation and determine if further adjustments to the model or training process are needed. This workflow provides a structured approach to handling image data, from initial preparation

Algorithm Used Encoder and Decoder

Encoders and decoders are fundamental components of many deep learning models, especially in tasks involving sequence-to-sequence (seq2seq) predictions, such as machine translation, image captioning, and text summarization. These architectures transform input data into a fixed-size representation and then decode this representation back into the desired output format.

Encoders

An encoder is a neural network that processes input data to produce a fixed-size representation, often referred to as a context vector or latent vector. The goal of the encoder is to capture the essential features and patterns of the input data in a compressed form. Architecture Encoders can take various forms depending on the type of input data. Recurrent Neural Network (RNN) Encoders: Commonly used for sequential data, such as text or time series. Variants include Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs). Convolutional Neural Network (CNN) Encoders: Typically used for image data, where convolutional layers extract spatial features. Transformer Encoders: Utilize self-attention mechanisms to process sequential data in parallel, capturing global dependencies efficiently.

Functionality

The encoder processes the input through a series of layers, each transforming the data into a higher-level representation. In RNNs, this involves processing the sequence step-by-step, updating hidden states. In CNNs, it involves applying convolutional filters to detect patterns. In transformers, it involves applying self-attention to capture relationships between all elements in the sequence.

Decoders

A decoder is a neural network that transforms the fixed-size representation produced by the encoder back into the desired output format. The decoder aims to reconstruct the original input data or generate a new output based on the encoded representation. Architecture: Similar to encoders, decoders can take various forms RNN Decoders: Generate sequences step-by-step, using the context vector from the encoder as initial input. CNN Decoders: Typically used in tasks like image generation or super-resolution, where deconvolutional layers reconstruct the spatial dimensions. Transformer Decoders: Use self-attention and encoder-decoder attention mechanisms to generate sequences in parallel. Functionality: The decoder starts with the context vector from the encoder and produces the output iteratively or in parallel. In seq2seq models, the decoder uses the context vector and generates the output sequence one element at a time, often employing techniques like beam search for improved performance. In CNN decoders, up sampling techniques like transposed convolutions or interpolation restore the original image size.

Transformer Models

The transformer model, introduced in the paper "Attention is All You Need" by Vaswani et al., revolutionized the field of deep learning, particularly in NLP. It employs self-attention mechanisms to handle dependencies in sequential data more effectively than RNNs. Architecture: The transformer consists of an encoder and a decoder, each composed of multiple layers of self-attention and feed-forward networks. Key components include. Self-Attention Mechanism: Allows the model to weigh the importance of different elements in the input sequence, capturing long-range dependencies efficiently. Positional Encoding Since transformers do not process data sequentially, positional encodings are added to input embeddings to provide information about the order of the sequence. Multi-Head Attention: Improves the model's ability to focus on different parts of the input by applying multiple attention mechanisms in parallel. Feed-Forward Network. Applies non-linear transformations to the output of the attention layers, enhancing the model's representational power. Encoder in Transformers The encoder consists of multiple identical layers, each containing A multi-head self-attention mechanism. A position-wise feed-forward network. Layer normalization and residual connections to stabilize training. Decoder in Transformers The decoder also consists of multiple identical layers, each containing A masked multi-head self-attention mechanism to prevent positions from attending to future positions during training.

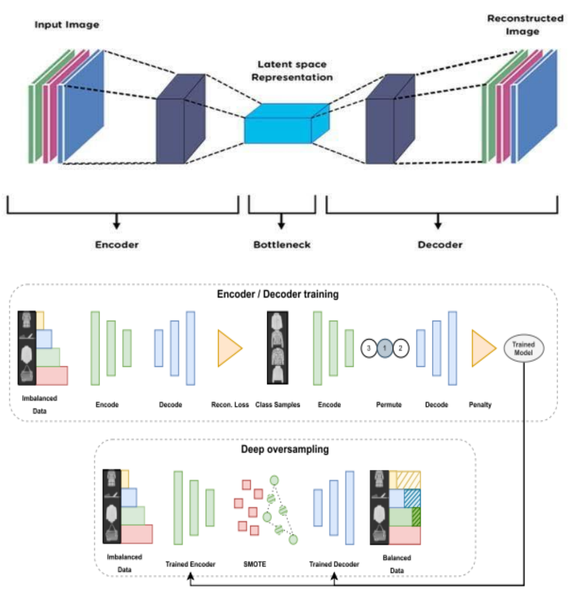


Figure 3.2 Transformer Models

3.5 System Design

Designing of system is the process in which it is used to define the interface, modules and data for a system to specified the demand to satisfy. System design is seen as the application of the system theory. The main thing of the design a system is to develop the system architecture by giving the data and information that is necessary for the implementation of a system.

3.5.1 Architecture Diagram

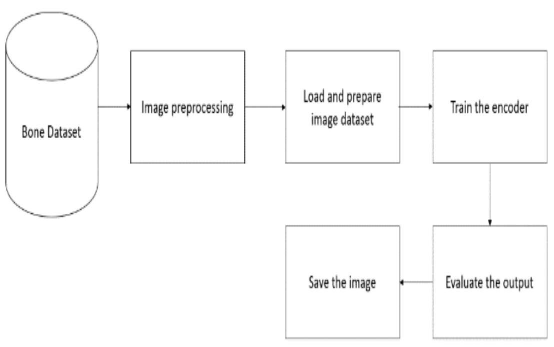


Figure 3.3 Architecture Diagram

3.5.2 Data Flow Diagram

Data flow diagrams are used to graphically represent the flow of data in a business information system. DFD describes the processes that are involved in a system to transfer data from the input to the file storage and reports generation.

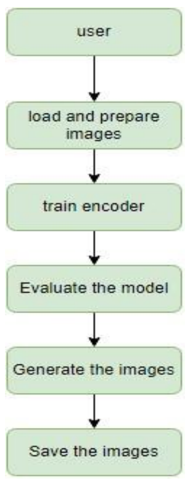


Figure 3.4 Data Flow Diagram

3.5.3 Use case Diagram

Use case diagrams are a way to capture the system's functionality and requirements in UML diagrams. It captures the dynamic behavior of a live system. A use case diagram consists of a use case and an actor.

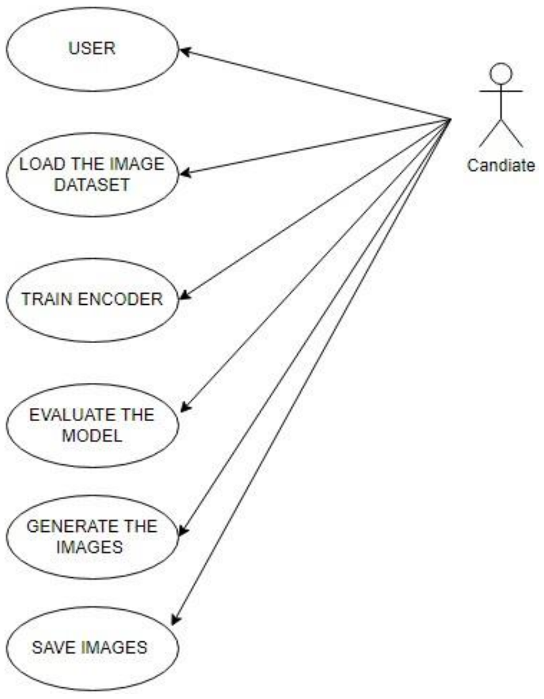


Figure 3.5 Use case Diagram

3.5.4 Class Diagram

Class diagrams are the main building block in object-oriented modeling. They are used to show the different objects in a system, their attributes, their operations and the relationships among them

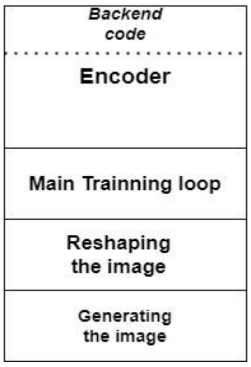


Figure 3.6 Class Diagram

3.5.5 Sequence Diagram

The sequence diagram of a system shows the entity interplay are ordered in the time order level. So, that it drafts the classes and object that are imply in the that plot and also the series of message exchange take place betwixt the body that need to be carried out by the purpose of that scenario.

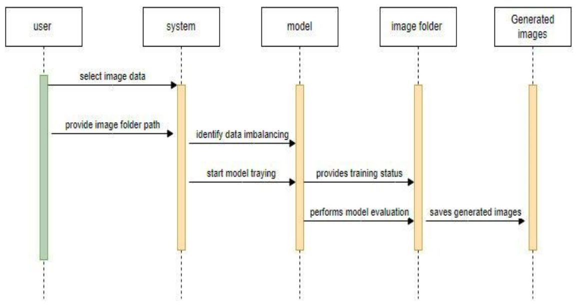


Figure 3.7 Sequence Diagram

3.5.6 State Flow Diagram

The below state chart diagram describes the flow of control from one state to another state (event) in the flow of the events from the creation of an object to its termination

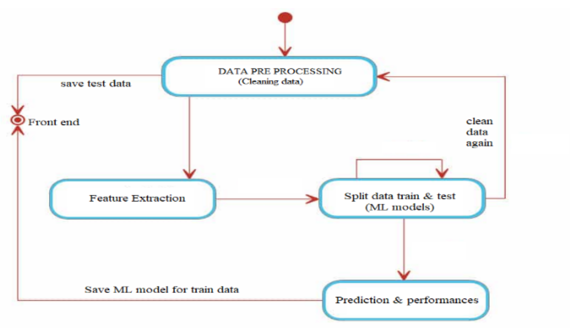


Figure 3.8 State Flow Diagram

3.5.7 Activity Diagram

Activity Diagrams describe how activities are coordinated to provide a service which can be at different levels of abstraction. Typically, an event needs to be achieved by operations, particularly where the operation is intended to achieve a number of different things that require coordination

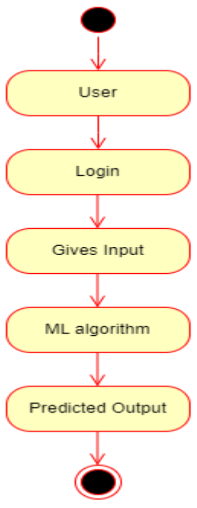


Figure 3.9 Activity Diagram

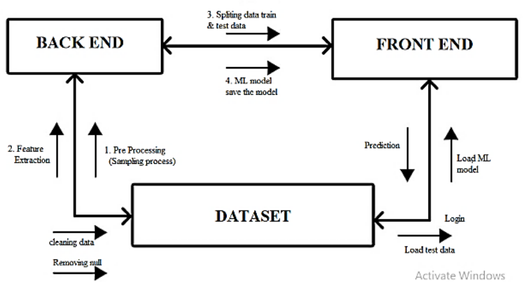


Figure 3.10 Colloboration diagram

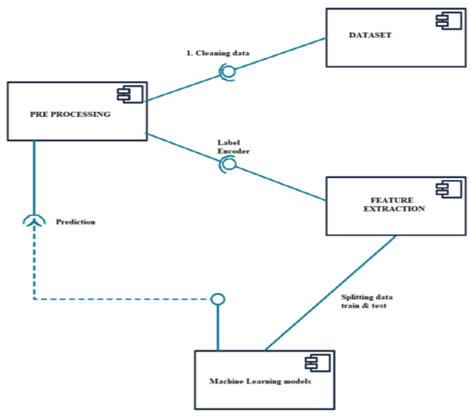


Figure 3.11 Component Diagram

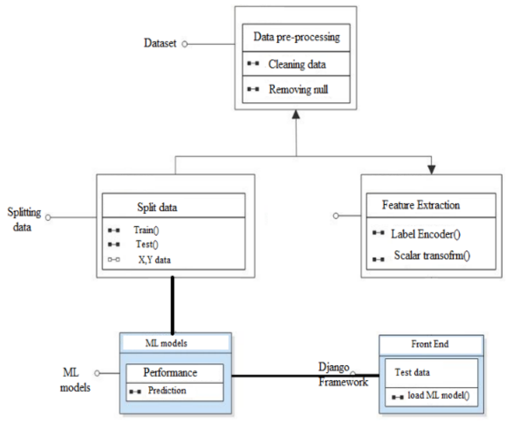


Figure 3.12 Object Diagram

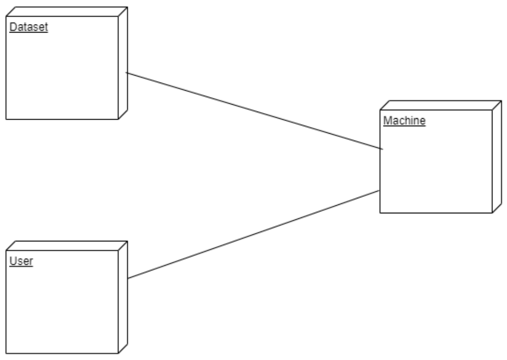


Figure 3.13 Deployement Diagram

4. Software Specification

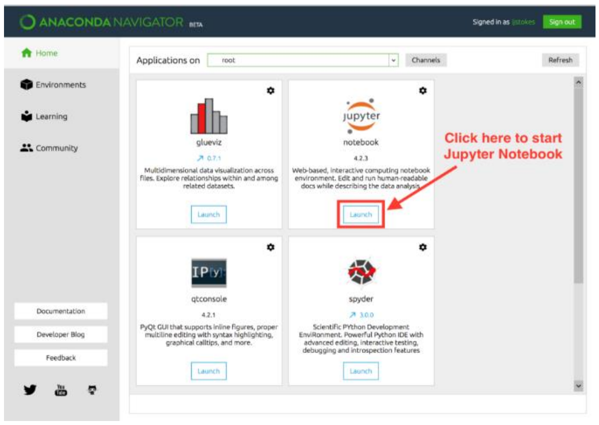
4.1 General

It is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

4.2 Anaconda

Anaconda distribution comes with more than 1,500 packages as well as the Conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator, as a graphical alternative to the Command Line Interface (CLI). The big difference between Conda and the pip package manager is in how package dependencies are managed, which is a significant challenge for Python data science and the reason Conda exists. Pip installs all Python package dependencies required, whether or not those conflict with other packages you installed previously. So, your working installation of, for example, Google Tensorflow, can suddenly stop working when you pip install a different package that needs a different version of the Numpy library. More insidiously, everything might still appear to work but now you get different results from your data science, or you are unable to reproduce the same results elsewhere because you didn't pip install in the same order.

Conda analyzes your current environment, everything you have installed, any version limitations you specify (e.g. you only want tensorflow>= 2.0) and figures out how to install compatible dependencies. Or it will tell you that what you want can't be done. Pip, by contrast, will just install the thing you wanted and any dependencies, even if that breaks other things.Open source packages can be individually installed from the Anaconda repository, Anaconda Cloud (anaconda.org), or your own private repository or mirror, using the conda install command. Anaconda Inc compiles and builds all the packages in the Anaconda repository itself, and provides binaries for Windows 32/64 bit, Linux 64 bit and MacOS 64-bit. You can also install anything on PyPI into a Conda environment using pip, and Conda knows what it has installed and what pip has installed. Custom packages can be made using the conda build command, and can be shared with others by uploading them to Anaconda Cloud, PyPI or other repositories.The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, you can create new environments that include any version of Python packaged with conda.



Multidimensional data visualization across files. Explore relationships within and among related datasets.

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

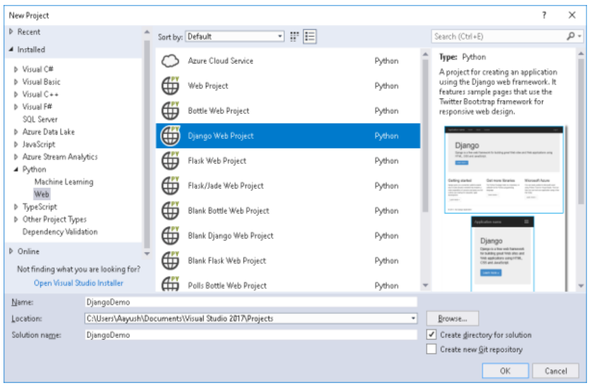
Figure 4.1 Anaconda Navigator

Anaconda Navigator is a desktop Graphical User Interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux.

The following applications are available by default in Navigator:

- JupyterLab
- Jupyter Notebook
- QtConsole
- Spyder
- Glueviz
- Orange
- Rstudio
- Visual Studio Code

Microsoft .NET is a set of Microsoft software technologies for rapidly building and integrating XML Web services, Microsoft Windows-based applications, and Web solutions. The .NET Framework is a language-neutral platform for writing programs that can easily and securely interoperate. There's no language barrier with .NET: there are numerous languages available to the developer including Managed C++, C#, Visual Basic and Java Script. The .NET framework provides the foundation for components to interact seamlessly, whether locally or remotely on different platforms. It standardizes common data types and communications protocols so that components created in different languages can easily interoperate. ".NET" is also the collective name given to various software components built upon the .NET platform. These will be both products (Visual Studio.NET and Windows.NET Server, for instance) and services (like Passport, .NET My Services, and so on). Microsoft VISUAL STUDIO is an Integrated Development Environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps.



Type: Python

A project for creating an application using the Django web framework. It features sample pages that use the Twitter Bootstrap framework for responsive web design.

Name: DjangoDemo
Location: C:\Users\Aayush\Documents\Visual Studio 2017\Projects
Solution name: DjangoDemo
[✓] Create directory for solution
[ ] Create new Git repository

Figure 4.2 Microsoft VISUAL STUDIO

Python is a powerful multi-purpose programming language created by Guido van Rossum. It has simple easy-to-use syntax, making it the perfect language for someone trying to learn computer programming for the first time. Python features are

- Easy to code
- Free and Open Source
- Object-Oriented Language
- GUI Programming Support
- High-Level Language
- Extensible feature
- Python is Portable language
- Python is Integrated language
- Interpreted
- Large Standard Library
- Dynamically Typed Language

4.3 PYTHON

- Python is a powerful multi-purpose programming language created by Guido van Rossum. It has simple easy-to-use syntax, making it the perfect language for someone trying to learn computer programming for the first time.

Features Of Python

1.Easy to code Python is high level programming language. Python is very easy to learn language as compared to other language like c, c#, java script, java etc. It is very easy to code in python language and anybody can learn python basic in few hours or days. It is also developer-friendly language.

2. Free and Open-Source Python language is freely available at official website and you can download it from the given download link below click on the Download Python keyword. Since, it is open-source, this means that source code is also available to the public. So you can download it as, use it as well as share it.

3. Object-Oriented Language One of the key features of python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, objects encapsulation etc.

4. GUI Programming Support Graphical Users interfaces can be made using a module such as PyQt5, PyQt4, wxPython or Tk in python. PyQt5 is the most popular option for creating graphical apps with Python.

5. High-Level Language Python is a high-level language. When we write programs in python, we do not need to remember the system architecture, nor do we need to manage the memory.

6. Extensible feature Python is a Extensible language. we can write our some python code into c or c++ language and also we can compile that code in c/c++ language.

7. Python is Portable language Python language is also a portable language. for example, if we have python code for windows and if we want to run this code on other platform such as Linux, Unix and Mac then we do not need to change it, we can run this code on any platform.

8. Python is Integrated language Python is also an Integrated language because we can easily integrated python with other language like c, c++ etc.

9. Interpreted Language Python is an Interpreted Language. because python code is executed line by line at a time. like other language c, c++, java etc there is no need to compile python code this makes it easier to debug our

code. The source code of python is converted into an immediate form called bytecode.

10. Large Standard Library Python has a large standard library which provides rich set of module and functions so you do not have to write your own code for every single thing. There are many libraries present in python for such as regular expressions, unit-testing, web browsers etc.

11. Dynamically Typed Language Python is dynamically-typed language. That means the type (for example- int, double, long etc) for a variable is decided at run time not in advance. Because of this feature we don't need to specify the type of variable.

Applications of Python

Web Applications

- You can create scalable Web Apps using frameworks and CMS (Content Management System) that are built on Python. Some of the popular platforms for creating Web Apps are: Django, Flask, Pyramid, Plone, Django CMS.
- Sites like Mozilla, Reddit, Instagram and PBS are written in Python.

4.3.1 Scientific and Numeric Computing

- There are numerous libraries available in Python for scientific and numeric computing. There are libraries like:SciPy and NumPy that are used in general purpose computing. And, there are specific libraries like: EarthPy for earth science, AstroPy for Astronomy and so on.
- Also, the language is heavily used in machine learning, data mining and deep learning.

4.3.2 Creating Software Prototypes

- Python is slow compared to compiled languages like C++ and Java. It might not be a good choice if resources are limited and efficiency is a must.
- However, Python is a great language for creating prototypes. For example: You can use Pygame (library for creating games) to create your game's prototype first. If you like the prototype, you can use language like C++ to create the actual game.

4.3.3 Good Language to Teach Programming

Python is used by many companies to teach rogramming to kids It is a good language with a lot of features and capabilities. Yet, it's one of the easiest language to learn because of its simple easy-to-use system.

4.4 Testing

White Box Testing is software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability and security. In white box testing, code is visible to testers so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing and Glass box testing. White box testing techniques analyze the internal structures the used data structures, internal design, code structure and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing or structural testing.

Working Process of White Box Testing

- Input: Requirements, Functional specifications, design documents, source code. Processing: Performing risk analysis for guiding through the entire process. Proper test planning: Designing test cases so as to cover entire code. Execute rinse-repeat until error-free software is reached. Also, the results are communicated. Output: Preparing final report of the entire testing process

Unit Testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program input produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Functional Test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals. Functional testing is centered on the following items. Valid Input identified classes of valid input must be accepted. Invalid Input: identified classes of invalid input must be rejected. Functions identified functions must be exercised. Output identified classes of application outputs must be exercised Systems/ Procedures interfacing systems or procedures must be invoked.

System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

Performance Test

The Performance test ensures that the output be produced within the time limits, and the time taken by the system for compiling, giving response to the users and request being send to the system for to retrieve the results.

Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects. The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level interact without error.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Acceptance testing for Data Synchronization

• The Acknowledgements will be received by the Sender Node after the Packets are received by the Destination Node

• The Route add operation is done only when there is a Route request in need

• The Status of Nodes information is done automatically in the Cache Updating process

5. IMPLEMENTATION

Python is a program that was originally designed to simplify the implementation of numerical linear algebra routines. It has since grown into something much bigger, and it is used to implement numerical algorithms for a wide range of applications.

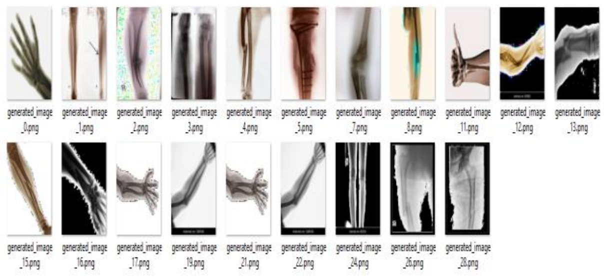


Figure 5.1 Output generated images

CONCLUSION

In this work, we introduced DeepSMOTE, a pioneering model designed to address the challenges of imbalanced data distributions in machine learning. By integrating the widely recognized SMOTE algorithm with advanced deep learning techniques, DeepSMOTE offers a robust solution for oversampling and balancing training datasets. This novel framework represents a significant advancement in handling class imbalance, particularly in the context of image data.DeepSMOTE stands out for its ability to operate directly on raw images, generate efficient low-dimensional embeddings, and produce high-quality synthetic images. These capabilities are enabled by a unique architecture that combines an encoder-decoder framework with SMOTE-based oversampling and an enhanced loss function. This integration allows DeepSMOTE to create artificial instances that effectively balance the training set, thereby mitigating bias and improving the performance of deep classifiers.Our extensive experimental evaluations demonstrate that DeepSMOTE surpasses existing state-of-the-art methods, including pixel-based and GAN-based oversampling algorithms. It not only excels in generating high-quality synthetic images but also exhibits remarkable robustness across various imbalance ratios, providing high model stability. The quality of the artificial images produced by DeepSMOTE is consistently superior, further validating its efficacy as an advanced resampling algorithm.

FUTURE WORK

Future work for the DeepSmote project could focus on several key areas to advance its capabilities and applications. Enhancing the algorithm itself could involve optimizing deep learning models, exploring hybrid approaches by combining DeepSmote with other oversampling techniques, and developing adaptive sampling methods to better handle varying levels of data complexity. To improve model training and evaluation, integrating advanced data augmentation techniques, expanding benchmarking to diverse datasets, and creating new metrics to assess synthetic data quality would be beneficial. Additionally, efforts could be directed toward improving computational efficiency through model optimization and developing scalable implementations for large datasets. Exploring practical applications in fields like healthcare and finance, while creating user-friendly tools for practitioners, will help extend DeepSmote's impact. On a theoretical level, understanding the model's behavior and generalization capabilities can provide deeper insights into its effectiveness. Engaging with the community through open-source development and educational resources will foster collaboration and ease of use. Finally, addressing ethical considerations by examining biases and ensuring transparency in synthetic data generation will promote fairness and trust in DeepSmote's applications.

REFERENCES

[1] C. Wu and H. Li, “Conditional transferring features: Scaling GANs to thousands of classes with 30% less high-quality data for training,” in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Glasgow, U.K., Jul. 2020, pp. 1–8.

[2] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic minority over-sampling technique,” *J. Artif. Intell. Res.*, vol. 16, no. 28, pp. 321–357, Jun. 2006.

[3] C. Huang, Y. Li, C. C. Loy, and X. Tang, “Deep imbalanced learning for face recognition and attribute prediction,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 11, pp. 2781–2794, Nov. 2020.

[4] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, “Spectral normalization for generative adversarial networks,” 2018, arXiv:1802.05957.

[5] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," 2016, arXiv:1606.03498.

[6] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of Wasserstein GANs," 2017, arXiv:1704.00028.

[7] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 214–223.

[8] M. Koziarski, "Radial-based undersampling for imbalanced data classification," *Pattern Recognit.*, vol. 102, Jun. 2020, Art. no. 107262.

[9] W.-C. Lin, C.-F. Tsai, Y.-H. Hu, and J.-S. Jhang, "Clustering-based undersampling in class-imbalanced data," *Inf. Sci.*, vols. 409–410, pp. 17–26, Oct. 2017.

[10] P. Vuttipittayamongkol and E. Elyan, "Neighbourhood-based undersampling approach for handling imbalanced and overlapped data," *Inf. Sci.*, vol. 509, pp. 47–70, Jan. 2020.

[11] G. Douzas and F. Bação, "Geometric SMOTE: A geometrically enhanced drop-in replacement for SMOTE," *Inf. Sci.*, vol. 501, pp. 118–135, Oct. 2019.

[12] H. He, Y. Bai, E. A. Garcia, and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, IEEE World Congr. Comput. Intell., Hong Kong, Jun. 2008, pp. 1322–1328.

[13] X. W. Liang, A. P. Jiang, T. Li, Y. Y. Xue, and G. T. Wang, "LR-SMOTE—An improved unbalanced dataset oversampling based on K-means and SVM," *Knowl.-Based Syst.*, vol. 196, May 2020, Art. no. 105845.

[14] Y. Yang, Q. Zhao, L. Ruan, Z. Gao, Y. Huo, and X. Qiu, "Oversampling methods combined clustering and data cleaning for imbalanced network data," *Intell. Autom. Soft Comput.*, vol. 26, no. 5, pp. 1139–1155, 2020.

[15] Y. Xu, X. Meng, Y. Li, and X. Xu, "Research on privacy disclosure detection method in social networks based on multi-dimensional deep learning," *Comput., Mater. Continua*, vol. 62, no. 1, pp. 137–155, 2020.

[16] M. Koziarski, B. Krawczyk, and M. Wozniak, "Radial-based oversampling for noisy imbalanced data classification," *Neurocomputing*, vol. 343, pp. 19–33, May 2019.

[17] M. Koziarski and M. Wozniak, "CCR: A combined cleaning and resampling algorithm for imbalanced data classification," *Int. J. Appl. Math. Comput. Sci.*, vol. 27, no. 4, pp. 727–736, Jan. 2017.

[18] K. Boonchuay, K. Sinapiromsaran, and C. Lursinsap, "Decision tree induction based on minority entropy for the class imbalance problem," *Pattern Anal. Appl.*, vol. 20, no. 3, pp. 769–782, Aug. 2017.

[19] D. Cieslak, T. Hoens, N. Chawla, and W. Kegelmeyer, "Hellinger distance decision trees are robust and skew-insensitive," *Data Mining Knowl. Discovery*, vol. 24, no. 1, pp. 136–158, 2012.

[20] F. Li, X. Zhang, X. Zhang, C. Du, Y. Xu, and Y.-C. Tian, "Cost-sensitive and hybrid-attribute measure multi-decision tree over imbalanced datasets," *Inf. Sci.*, vol. 422, pp. 242–256, Jan. 2018.

[21] S. Datta and S. Das, "Multiobjective support vector machines: Handling class imbalance with Pareto optimality," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 5, pp. 1602–1608, May 2019.

[22] Q. Fan, Z. Wang, D. Li, D. Gao, and H. Zha, "Entropy-based fuzzy support vector machine for imbalanced datasets," *Knowl.-Based Syst.*, vol. 115, pp. 87–99, Jan. 2017.

[23] K. Qi, H. Yang, Q. Hu, and D. Yang, "A new adaptive weighted imbalanced data classifier via improved support vector machines with high-dimension nature," *Knowl.-Based Syst.*, vol. 185, Dec. 2019, Art. no. 104933.

[24] Q. Dong, S. Gong, and X. Zhu, "Imbalanced deep learning by minority class incremental rectification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 6, pp. 1367–1381, Jun. 2019.

[25] Y.-H. Liu, C.-L. Liu, and S.-M. Tseng, "Deep discriminative features learning and sampling for imbalanced data problem," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Singapore, Nov. 2018, pp. 1146–1151.

[26] P. Wang, F. Su, Z. Zhao, Y. Guo, Y. Zhao, and B. Zhuang, "Deep class-skewed learning for face recognition," *Neurocomputing*, vol. 363, pp. 35–45, Oct. 2019.

[27] C. Cao and Z. Wang, "IMCStacking: Cost-sensitive stacking learning with feature inverse mapping for imbalanced problems," *Knowl.-Based Syst.*, vol. 150, pp. 27–37, Jun. 2018.

[28] S. H. Khan, M. Hayat, M. Bennamoun, F. A. Sohel, and R. Togneri, "Cost-sensitive learning of deep feature representations from imbalanced data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 8, pp. 3573–3587, Aug. 2018.

[29] C. Zhang, K. C. Tan, H. Li, and G. S. Hong, "A cost-sensitive deep belief network for imbalanced classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 1, pp. 109–122, Jan. 2019.

[30] D. Devi, S. K. Biswas, and B. Purkayastha, "Learning in presence of class imbalance and class overlapping by using one-class SVM and undersampling technique," *Connection Sci.*, vol. 31, no. 2, pp. 105–142, 2019.