Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Generative Adversarial Networks for Improving Imbalanced Classification Performance

Snehal Bhatia
Supervisor: Prof. Rozenn Dahyot

August, 2019

A Dissertation submitted to the University of Dublin,
in partial fulfilment of the requirements for the degree of
Master of Science in Computer Science
(Future Networked Systems)

Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism ‘Ready Steady Write’, located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: ____________________ Date: ____________________

Abstract

The advent of deep learning has resulted in extremely powerful classification models which have led to significant progress in many domains of computer vision such as object recognition and image classification. However, most existing classifiers assume the underlying training dataset to be evenly distributed. In real-life datasets, it is often observed that some of the classes have far less quantity of data samples than the others. An example of this could be a dataset for animal detection collected from a wildlife sanctuary where very few animals belonging to 'endangered species' class were present. Due to the underrepresentation of the minority classes, the data samples belonging to them often pass as noise or outliers, or are ignored, ultimately resulting in their misclassification. More often than not, the correct detection of minority class instances is of the utmost importance. Therefore, we study the detrimental effects of class imbalance on the classification performance of the Hybrid CNN-SVM architecture, and aim to introduce measures to overcome this issue. We artificially introduce class imbalance in two benchmark datasets, FMNIST and CIFAR-10, and observe that the classification Accuracy and F1-Score both drop by an average of 7%, when compared to the performance on the original dataset. To combat this problem, we use data augmentation strategies to re-balance the datasets. We first explore the traditional data augmentation practices of applying geometric and photo-metric transformations on the existing images of minority classes, such as image rotation, scaling, zooming, blurring, whitening, shearing, etc. Then, we propose the use of a modified architecture of Generative Adversarial Networks (GAN), called Wasserstein GAN with Gradient Penalty (WGAN-GP) to generate new data samples. Since their introduction in 2014, GANs have shown immense potential in mimicking data distribution and generating realistic images using low amounts of training data. Training the classifier on the datasets augmented using WGAN-GP, we observe an average increase of 4% in the classification Accuracy and F1-Score for FMNIST dataset, and this is even more for the more complex CIFAR-10 dataset, where a 6.2% improvement is achieved. This is significantly better than the improvement achieved using the baseline method of dataset augmentation using image transformations, and it has proven to be a more promising solution for real-world datasets which are becoming increasingly complex and diverse.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Professor Rozenn Dahyot, who has not only provided immense support, guidance and feedback through each phase of this dissertation, but has also been a constant inspiration as a researcher. The inputs given by Professor Stephen Barrett have also been invaluable for structuring and completing my dissertation.

I am grateful to Trinity College Dublin and the School of Computer Science and Statistics for helping me build a solid foundation of computer science principles as well as research methods, and providing me with a platform to excel.

The moral support and motivation extended to me by my family and friends have been indispensable factors in the completion and success of this dissertation, and for that I am truly thankful.

Any omission of acknowledgement does not reflect my lack of regard or appreciation.

SNEHAL BHATIA

University of Dublin, Trinity College Dublin

August 2019

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview

In recent times, classification algorithms have significantly advanced with deep learning, and have achieved near-perfect results in the fields of computer vision, natural language processing, medical sciences, cyber security, etc. However, the performance of supervised deep learning based classifiers is hugely dependent on not only the quantity but also the quality of the data samples available to learn from.

While the data pre-processing steps for machine learning and classification algorithms mostly take care of the sufficiency of the total amount of data available, noise in the dataset, missing values and lack of standardization, one factor that is often overlooked is the skew in data distribution, or class imbalance. This situation where some classes of the datasets have significantly higher number of data instances than other classes handicaps the classification algorithm's ability to correctly classify a minority class data sample, since it becomes biased towards the majority classes during the learning process.

Imbalance in real-world datasets is prevalent as the occurrence of exceptional events is rare, for example, natural disasters such as Tsunami. With internet-of-things permeating our everyday life through smart and connected objects (phones, electronic devices, etc.) and the emergence of highly-connected smart cities, the sources of data collection have become highly dispersed and heterogeneous in nature, giving rise to complex unevenness in data sample distribution. For example, a weather forecasting application could be collecting data from not only physical sensors but also incorporating sentiment analysis of social media posts [1].Since such applications often operate in real-time, it is hard to determine if some of the sensors would fail at a given time and hence introduce a skew in the expected data distribution. Another example in this domain could be the advance prediction of traffic congestion using a dataset containing images captured by traffic cameras in a city where such situations are rare [2].Data loss and corruption due to human errors or technical glitches can also contribute to this problem.

Learning from imbalanced data is an important problem for the research community as well as industrial practitioners [3].and has found many uses in information retrieval systems, fraud detection, detection of oil spills in radar images, telecommunications, bioinformatics, medical diagnosis, detection of rare particles in high-energy physics, etc. [4], [5], [6], [7], [8], [9]. Therefore, in this research, we [1] have studied the effects of class imbalance on the performance of a hybrid CNN (Convolutional Neural Network) and SVM (Support Vector Machine) classifier on two multi-class datasets of varying complexity, FMNIST [10] and CIFAR-10 [11] by artificially introducing imbalance in them. Further, data augmentation has been explored as a measure to combat the detrimental effects of this imbalance. We first experiment with creating new data samples for the minority classes by applying various geometric and photo-metric transforms such as image rotation, flipping, zooming, blurring, shearing, whitening, etc. Then, we introduce a variation of Generative Adversarial Networks (GANs) [12] as an image-generation tool, to expand the data quantity of the imbalanced classes.

GANs have recently achieved remarkable success in many areas of computer vision, and are essentially composed of two competing neural networks, called the generator and the discriminator. The generator learns the feature space of a dataset and uses it to generate new images which are similar to the real ones in the existing data. The discriminator, on the other hand, acts as a classifier which can distinguish a real image from an artificially generated image. When operating in tandem, these help augment the dataset with new, realistic-looking images. Since the original GAN architecture (also called "Vanilla GAN") has certain drawbacks in terms of training efficiency and convergence, we have explored a developed version called Wasserstein GAN with Gradient Penalty (WGAN-GP) to achieve better results.

The aim of this research is to establish that an imbalanced multi-class dataset deteriorates the performance of supervised classification, and that it can be overcome by using data augmentation methods. Further, we aim to compare the efficiency of the benchmark image data augmentation strategy of geometric and photo-metric transformations with our method of using WGAN-GP. The implementation has been done using Python 3 language [13] and Keras library [14].

1.2 Motivation

The detrimental effect of imbalanced classes in datasets on the performance of a classifier can often go unnoticed, leading to not only misclassifications, but in some cases also to a false sense of success. One such example could be a dataset containing information

[1]Since this research has been carried out in collaboration with my supervisor, Professor Rozenn Dahyot, the pronoun "we" has been used wherever applicable.

regarding 100 patients of a hospital, out of which 99 are healthy and only one has cancer. A classifier trained on this dataset may label the patient suffering from cancer as healthy, and still achieve an accuracy of 99 %. If we extrapolate this scenario to a dataset containing a total of 1 million samples, it may possibly lead to 10,000 cancer-affected patients being labelled as cancer-free, which highlights the gravity of the situation. Such classifiers are useless to us, as the minority cases are often the most important to identify. Even though the accuracy illusion mentioned above can be corrected using other metrics to evaluate classifier performance, the fact is that minority class instances still remain likely to be wrongly classified.

I first noticed this problem of skewed data distributions in open-source datasets made available by the governmental departments of developing smart cities, while working with the Noise Pollution Dataset gathered and monitored by Dublin City Council. This prompted me to delve deeper into this area of research, and I noticed the significance of image classification in smart city applications such as road sign detection for self-driving cars, face detection of wanted criminals using city cameras, etc. Therefore, I have directed my research towards the study of effects of class imbalance on image datasets. The recent success of Generative Adversarial Networks in various domains such as improving image resolution, face synthesis, etc. motivated me to leverage their superior image generation capabilities from limited available training data to create new data samples for classes which are in minority in a dataset, to achieve a balanced distribution and ultimately improve the performance of image classifiers.

1.3 Dissertation Structure

This dissertation is organized into six chapters, starting with the Introduction as Chapter 1, which introduces the research question, the significance of the problem addressed and the motivations for exploring this area of research. It is followed by the Background Research in Chapter 2 which explores the underlying concepts used throughout this study, as well as a comprehensive literature review of the state of the art. The document then progresses into the Design approach adopted for the study in Chapter 3, outlining the various trade-offs and factors considered while making the design decisions. These concepts from the design are then actualized in Chapter 4 which provides specific Implementation details, as well as the Experiments that have been conducted in order to validate our hypotheses. The results obtained are then analysed and discussed in Chapter 5, and the insights obtained are then interpreted into an answer for the research question posed. The final chapter concludes all our findings and highlights some of the limitations of the research, with suggestions for improvement and future work to extend this study.

2 Background Research

This chapter details the underlying concepts and techniques used in this study. It also presents a thorough review of the state of the art, based on which further research has been done. The first section talks about image classification and the various architectures employed for it, followed by a section on the study of dataset imbalance and methods to overcome its effects. Lastly, we describe the structure of GANs and their evolution into WGAN-GP, which is used as a data augmentation tool.

2.1 Image Classification

Image classification is a complex task, which consists of a pipeline of sub-tasks such as image pre-processing, object detection and segmentation, feature extraction and assigning labels to objects, i.e., their classification. Classification tasks can be either 'Supervised' or 'Unsupervised' in nature. In the former methodology, the possible set of classes is known prior to the training process and labelled training data is used, whereas in the latter, the classes are not known in advance.

In general, the usual method of classifying objects can be observed in Figure 2.1. The first module is a feature extractor which transforms the input raw images into feature vectors. These feature vectors and class categories extracted from a dataset are fed into a classifier to train it for predicting the class labels or scores of any input image.

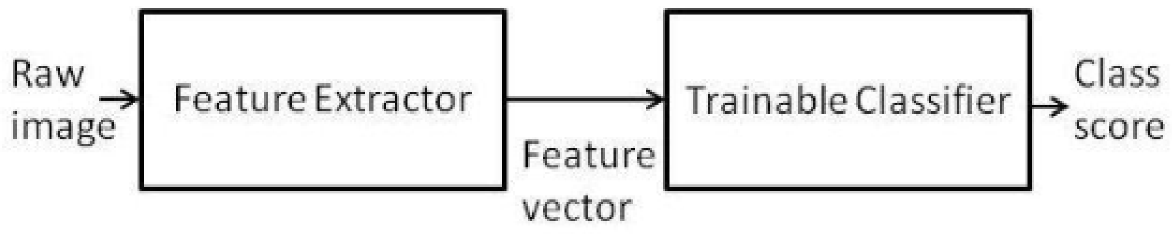


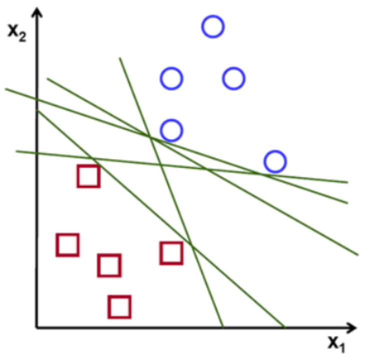
Figure 2.1: General Image Classification Workflow [15]

In this section, we shall discuss some popular classification architectures, which we will use in our research.

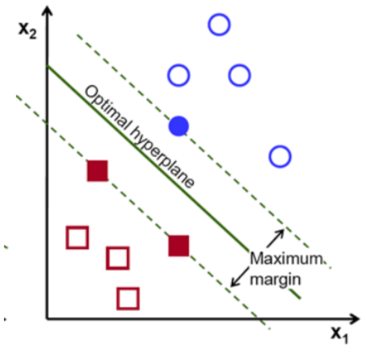
2.1.1 Support Vector Machine

Support vector machines (SVMs) are supervised learning models which construct a set of hyperplanes or decision boundaries in an N-dimensional space (N being the number of features), with the objective of converting a non-linear separable problem into a linear separable problem by finding the hyperplane that distinctly classifies the data points. Such a hyperplane has the maximum distance between data points of both classes. The data points closest to a hyperplane are called "support vectors", and influence the orientation as well as the position of the hyperplane [16].

As can be observed in Figure 2.2 .Maximizing this margin provides reinforcement, to enable the future data points to be classified with more confidence, and reducing the empirical risk of misclassifications [17]



(a) Set of possible Hyperplanes



(b) The Optimal Hyperplane

Figure 2.2: Selecting the Optimal Hyperplane in a 2D Feature Space [16]

SVMs use a kernel function to project the input training data to a feature space of higher dimension, resulting in a linearly separable dataset. Often in high-dimension feature spaces, overfitting is observed in classification tasks, however, in SVMs, it is controlled through structural risk minimization [18].

By design, SVMs are binary classifiers. They can be extended to work with multiple classes, and are called Multiclass Support Vector Machines [19]. This can be achieved by either combining several binary classifiers, or by directly considering all of the data in one optimization formulation.

2.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs)[1] are the most popular deep-learning architectures, which have a powerful ablility of feature extraction and thus find their use in diverse fields, including image recognition and classification.

In general, a CNN architecture is composed of four types of layers [21] which can also be observed in Figure 2.3.

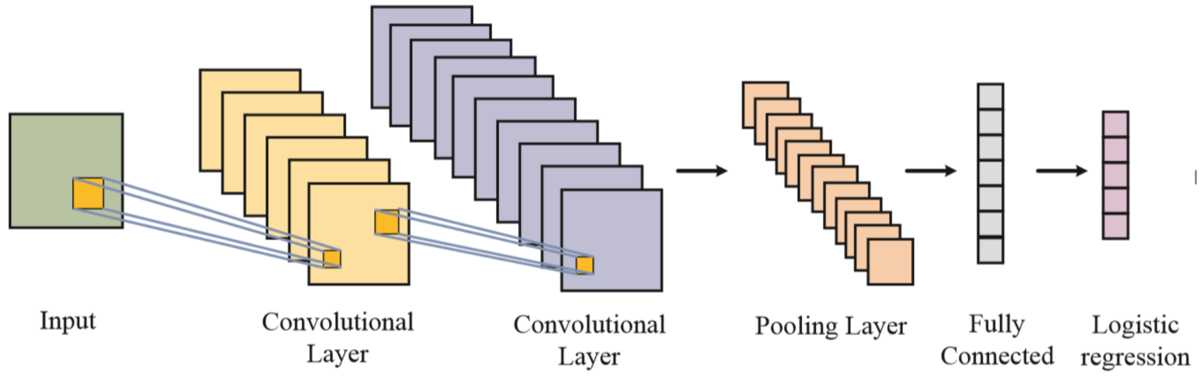


Figure 2.3: Structure of a Sample CNN [21]

Convolutional Layer

Convolutional Layers of a CNN serve as feature extractors in the architecture. They are composed of multiple convolution kernels[2], whose parameters need to be learned in the training process. Each filter gets convoluted with the input, to compute an activation map which is comprised of neurons. Each neuron in the activation map is only connected to a small local region of the input, which allows the network to learn locally optimized filters which can leverage the local spacial correlation between pixels and extract more relevant features.

[1]Convolutional Neural Networks have emerged from Feed-Forward Neural Networks or Multi-Layer Perceptrons. They typically consist of an input layer and an output layer, along with a variable number of hidden layers. They make use of activation functions in the hidden layers, and the back-propagation algorithm for computing the gradient values and updating the weights and biases in the network. For more details, refer to Chapter II of the Deep Learning handbook [20].

[2]In image processing, convolution is the operation of adding each element (pixel) to its local neighbours, weighted by a kernel (matrix).

Pooling Layer

This layer usually connects the previous convolutional layer, and is constructed with the aim of reducing the resolutions of feature maps to aggregate the input features. Some of the most commonly used pooling operations are max pooling and average pooling.

Fully Connected Layer

This layer is generally placed between pooling layer and logistic regression layer with the aim of sending the learned distributed feature representation to one space so that high-level reasoning can be performed. All neurons of a layer are connected to every single neuron of the next layer.

Logistic Regression Layer

It is the last layer of a CNN and is widely used in multi-class classification architectures. The softmax function is usually employed as the activation function of this layer, which predicts the probability of a given sample belonging to a class, taking into account the sampled data and the learned weights in the model.

In the testing phase, each data sample is assigned a probability value for each class, and it is ultimately classified into the class which it had the highest probability for.

2.2 The Class Imbalance Problem

Supervised classification algorithms work with labelled training datasets to predict which class a given (unseen) sample may belong to. In real-life applications, it is often the case that in the training dataset, some classes have significantly higher number of data samples than others. This has been observed in many domains, ranging from computer vision [22], medical diagnosis [23], fraud detection [24], computer and networks security [25],etc. Broadly, the two forms in which imbalance can be observed in datasets are [26]:

- *Intrinsic Imbalance*, which is due to the natural frequency of occurrence of data , and
- *Extrinsic Imbalance*, which is introduced through external factors such as data collection and storage methods

A consequence of dataset imbalance problem is that standard classification learning algorithms are often biased towards the majority class(es) and the data samples belonging to minority class(es) get easily misclassified . As there is a genuine lack of data due to the infrequency of occurrence of some events, for example, in detecting nuclear leaks, it is crucial to be able to learn from extremely underrepresented classes as they represent the most important concepts. In most cases, binary classification problems arise with imbalanced datasets, however, multi-class problem occurrences are also not rare, and are also more difficult to solve due to the presence of several minority classes [27, 28].

It has been proven that class imbalance has a significant detrimental effect on the training of classifiers on traditional machine learning classifiers [29], multi-layer perceptrons (MLPs) [30], as well as deep-learning based classifiers [31]. There are several possible reasons for a well-performing classification algorithm to not necessarily achieve the same performance for an imbalanced dataset [32]:

- Standard performance measures such as accuracy are used for guiding the learning process. However, accuracy is not a proper measure in case of imbalanced datasets as it does not distinguish between the number of correctly classified data instances for the different classes
- When the minority class samples are very small in number, they may be incorrectly identified as noise and subsequently be discarded by the classifier. However, we also need to consider the fact that the presence of few real noisy data points may also cause a degradation in minority class identification as the number of training samples available for such classes are very low
- The classification rules to predict a minority class are highly specialized, and may be discarded in favour of the generic rules, or the rules which are used to predict the majority class samples

Deep Learning models in specific use gradient descent optimization to adjust the weight parameters of the network to minimize the loss function, or the error between expected and actual output. It has been shown that in imbalanced datasets, the majority class dominates the net gradient which is responsible for updating the model's weights [33] which increases the error for the minority class, causing the network to get stuck in a slow convergence mode.

2.2.1 Measure of Dataset Imbalance

A common method to represent the imbalance in a dataset is through the maximum between-class-imbalance level, which can be expressed as a ratio, ρ , between the maximum and minimum class size, over all i classes. If C_i is the number of samples in class i, then:

$$\rho = \frac{\max\{|C_i|\}}{\min\{|C_i|\}} \qquad (1)$$

As an example, consider a dataset containing 1 million samples, of which only 1% represent the minority class. In this case, ρ=100, which is very high. However, when we look at the actual number of samples, we can observe that we have 10,000 samples from the minority class, which still may be enough to train a classifier. Therefore, the total number of minority samples available is of more significance than the percentage of minority.

2.2.2 Methods of Handling Class Imbalance

n order to remove the bias towards the majority classes in a dataset, there are various modifications that can be made on the data as well as on the classification algorithms. These methods can be of two types [34], as detailed below:

Data Level Methods

In this approach, the class distribution in the training dataset is changed in order to decrease the imbalance and reduce noise (such as mislabelled samples), so that standard classification algorithms can perform well on them.

Data Level Methods involve resampling of the data, until all classes in the data are equally represented. In general, they are of two types (Figure 2.4):

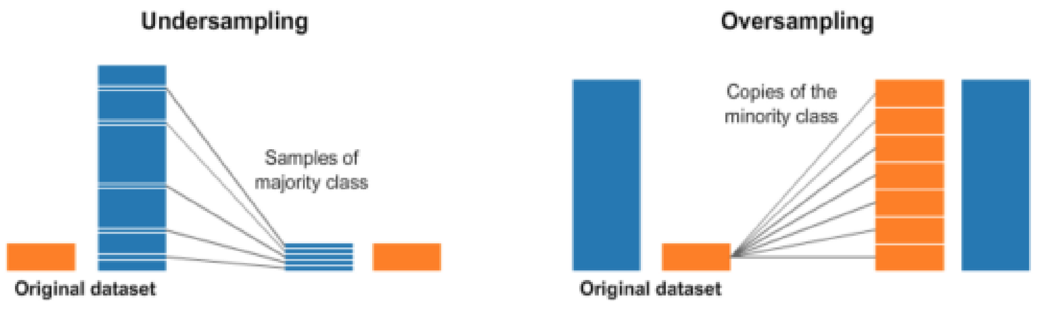


Figure 2.4: Data Level Class Balancing Methods: Undersampling and Oversampling [35]

A. Random Oversampling (ROS):
It is one of the most commonly used methods in deep learning [36, 37, 38]. It simply involves the random replication of the minority samples to achieve a balance.

Although over-sampling leads to increased training time due to increase in training dataset, and may cause over-fitting[3] in some cases [3], it is a widely suggested method with deep learning [31, 39, 40] , especially for multi-class image data, where over-sampling to the level of class data works best [31, 41],Further, it has also been shown that oversampling does not cause the over-fitting of convolutional neural networks, as compared to some classical machine learning models [31].

B. Random Under Sampling (RUS):
This approach involves randomly discarding samples from the majority classes until each class has similar number of samples.

An obvious disadvantage of this method is that it results in reduction of the total amount of information that a model has to learn from.

However, in some situations, especially in big data applications, it may be preferable to have a smaller dataset to reduce training time, and when done in an informed fashion, can be a preferable approach [42].

Algorithm Level Methods

Algorithmic methods for reducing the effect of imbalance in datasets do not alter the training distribution. Instead, adjustments are made to the learning process in a manner that increases the importance of the minority class. This is done by introducing penalties, shifting the decision threshold to reduce the bias towards the majority classes or assigning costs and weights in a manner that is favourable to the under-represented class.

However, making such modifications to classification algorithms requires extensive domain knowledge and problem-specific expertise to determine empirical factors, or can be alternatively achieved via trial and error. Such an approach may be expensive, and even impractical in some cases, and cannot be generalized to a variety of scenarios.

2.2.3 Data Augmentation using Image Transformations

Data Augmentation is the process of generating samples by applying random transformations on the existing training data to improve the performance of classification models [43]. Small affine transformations of data can preserve the labels of an image [44] and improve the generalization error.

[3]Overfitting is the case where the model fits too closely to the training data and as a result is unable to generalize to new data.

Some of the most common geometric image transformation strategies are rotation, translation, flipping, scaling, shearing, etc., and can be observed in Figure 2.5. These transformations are applied on every pixel *(x,y)* of an image to obtain new pixels *(x', y')*. As an example, *x'=x+2* and *y'=y-3* would shift the image left by 2 pixels and down by 3 pixels, whereas *x'= -y* and *y'=x* would rotate the image clockwise by a 90 degree angle.

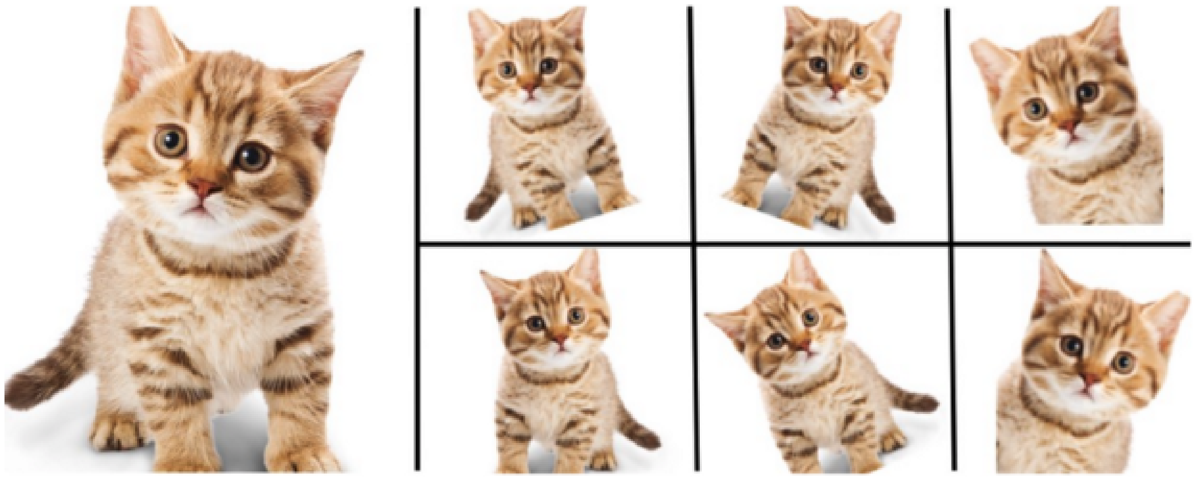
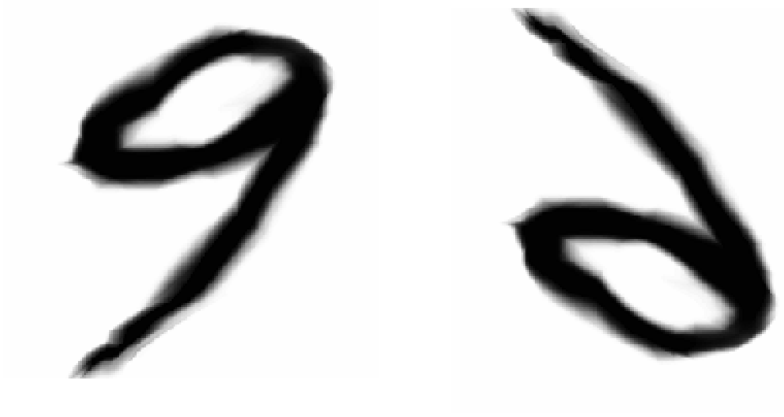


Figure 2.5: Original Image (Left) and Transformed Images (Right) [45]

On the other hand, photo-metric transformations like blurring, whitening, etc. are applied by changing the intensity and colour values of each pixel, which can be achieved by the use of various filters and convolution operations. The details of the methods used by us will be presented in Chapters 3 and 4.

Even though these methods are the most intuitive and universally used, data augmentation may involve many manual choices, which if made incorrectly, may result in uninformative samples or detrimental effects on classification. While repetitiveness of data may result in over-fitting of the classification model, it is also possible to produce incorrectly labelled data, as illustrated in Figure 2.6.



(a) Original Image (b) Flipped Image

Figure 2.6: Flipping Image (a) having class label "nine" from the MNIST dataset [46] would result in (b), which should actually be labelled as class "six", but in this case would be incorrectly categorized as "nine", causing confusion for the model

2.3 Generative Adversarial Networks

The Generative Adversarial Network (GANs) framework was first introduced for artificially generating realistic images from scratch [12]. Since then, GANs have been employed for a variety of image processing and computer vision tasks, such as generating high resolution images from low resolution input images [47], texture synthesis in images [48], human face synthesis, etc [49].

This generative capacity of GANs make them suitable for the purpose of data augmentation, and many recent studies have shown to tackle the problem of imbalanced datasets in classification by using variations of the GAN architecture.

This section introduces the architecture of Generative Adversarial Nets as a tool for training data augmentation, and some proposed modifications to enhance their performance.

2.3.1 Vanilla GAN

The original architecture introduced by Ian Goodfellow et al [12] has been dubbed as "Vanilla GAN". In this section we would be referring to it simply as "GAN"..

A GAN (Figure 2.7) consists of two sub-networks, which are competing Artificial Neural Networks (ANNs), namely the 'Generator' ($\boldsymbol{G}$) and the 'Discriminator' ($\boldsymbol{D}$). The Generator learns the distribution of input data or from a sample noise distribution and generates new

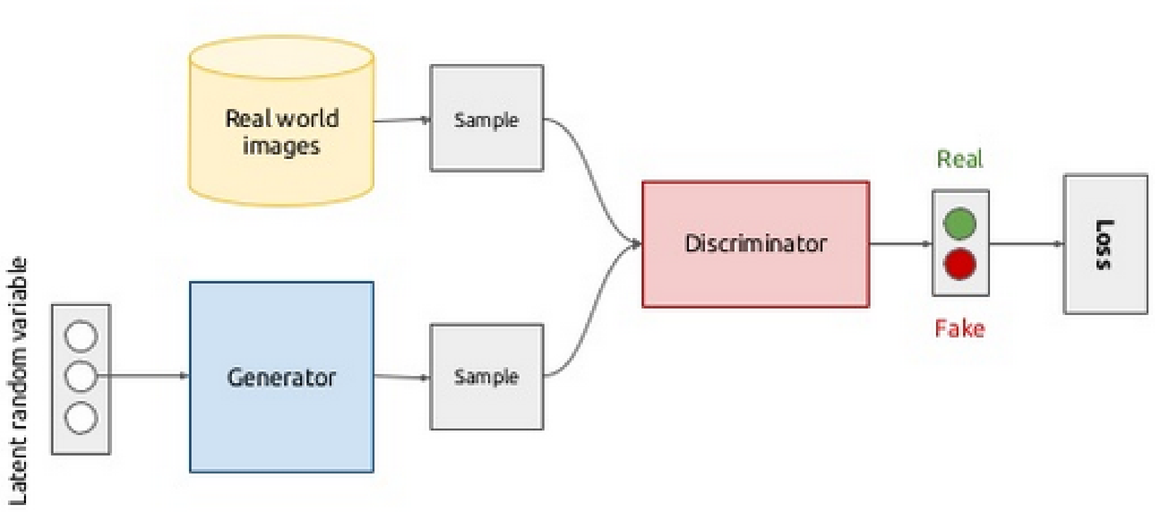


Figure 2.7: The Conceptual Model of Generative Adversarial Network [50]

data which is as realistic as possible. At the same time, the Discriminator, which is essentially a binary classifier, is trained to distinguish between the real data points and artificially generated data points (by the generator). Note that the GAN's end goal is to generate the relevant image, given a label.

A GAN is often defined as a two-player minimax game [12] in which G wants to minimize the value function[4] V, and D aims to maximize it. The next section details the value function for Vanilla GAN.

Value Function for Vanilla GAN

To explain the value function of the Vanilla GAN, certain terms need to be defined:

- z: Noise vector
- x: Training Sample (or x_{real})
- $G(z)$:The output of the Generator G, or x_{fake}
- $D(x)$: Discriminator's output for x_{real}. It takes the value of a probability function in the range $P(y|x_{real}) \rightarrow \{0,1\}$
- $D(G(z))$: Discriminator's output for x_{fake}. It takes the value of a probability function in the range $P(y|x_{fake}) \rightarrow \{0,1\}$

[4]The terms "value function", "loss function", "cost function" and "objective function" have been used interchangeably in this study

The discriminator model should be such that it maximizes the score for real data (***D(x)***) and minimizes that of fake data ***D(G(z))***. Therefore, at ***D***:

$$D_{loss_{real}} = \log(D(x)) \tag{2}$$

The Equation (2) is a part of the overall loss function which when maximized, optimizes the discriminator to recognize real images better.

On the other hand, minimizing the value in Equation (3) below optimizes the discriminator to recognize the generated images better.

$$D_{loss_{fake}} = \log(1\text{-}D(G(z))) \tag{3}$$

Now,

$$D_{loss} = D_{loss_{real}} + D_{loss_{fake}} \tag{4}$$

Therefore, using Equations (2), (3) and (4), the total value function for discriminator can be defined as Equation (5).

$$D_{loss} = \frac{1}{m}\sum{}_{\mathrm{i=1}}{}^{\mathrm{m}} \log(D(x^{\mathrm{i}})) + \log(1 - D(G(z^{\mathrm{i}}))) \tag{5}$$

Similarly, the generator model should maximize the score assigned to the fake data(***D(G(z))***). Therefore, at ***G***:

$$G_{loss} = \log(1 - D(G(z))) = -\log(D(G(z))) \tag{6}$$

Using Equation (6), the total value function for ***G*** can be defined as:

$$G_{loss} = \frac{1}{m}\sum{}_{\mathrm{i=1}}{}^{\mathrm{m}} \log(1 - D(G(z^{\mathrm{i}}))) \tag{7}$$

Combining (5) and (7) we get the GAN value function (Equation (8)):

$$min_{\mathrm{G}}\, max_{\mathrm{D}}\, V(D,G) = E_{\,\mathrm{x}\in\ \mathrm{p_{data}(x)}}[\log D(x)] + E_{\,\mathrm{z}\in\ \mathrm{p_z(z)}}[\log(1 - D(G(z)] \tag{8}$$

In Equation (8), the sum of the two expectations or mean values essentially represents the Jensen-Shannon Divergence[5] between the real data and the fake data generated.

[5] Jensen-Shannon Divergence is a measure of similarity between two probability distributions, bounded by

GAN Training Process

The training process of a GAN can be viewed as a double feedback-loop [51] where the discriminator is in a feedback loop with the real images and the generator uses the feedback from the discriminator to learn how to produce images that are realistic enough to fool the discriminator. The feedback process is repeated throughout the training phase, until Nash Equilibrium is achieved[6]. This process is called **Adversarial Training**.

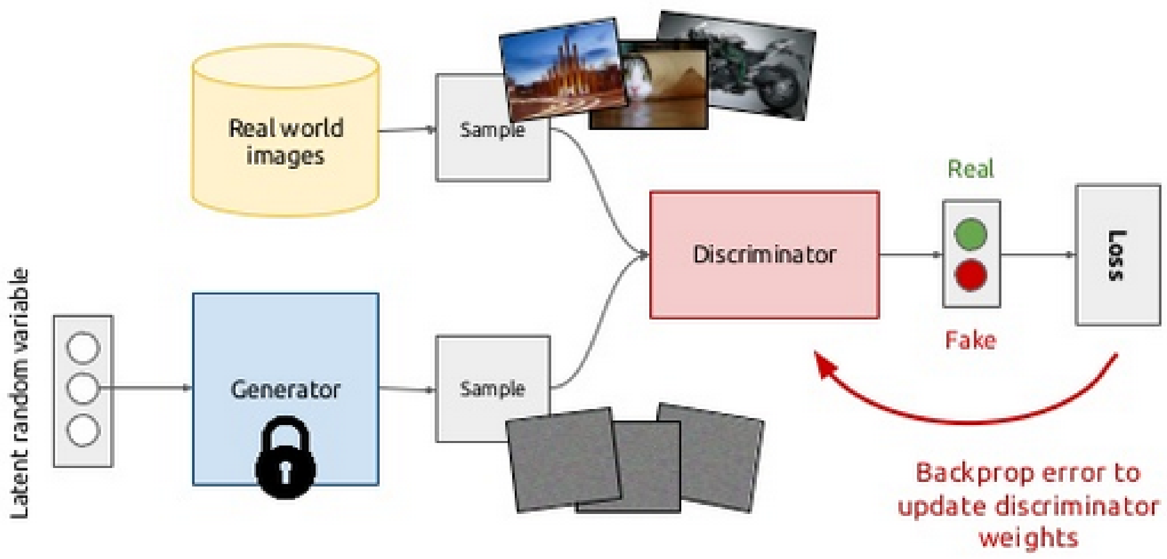


Figure 2.8: Discriminator Training Process [50]

[0,1]. More details can be found at: https://en.wikipedia.org/wiki/JensenShannon_divergence

[6]In Game Theory, when multiple interacting, non-cooperating participants are involved, Nash Equilibrium is the state of stability achieved when no participant can benefit solely from changing its own strategy or actions if the other players' strategies remain constant.

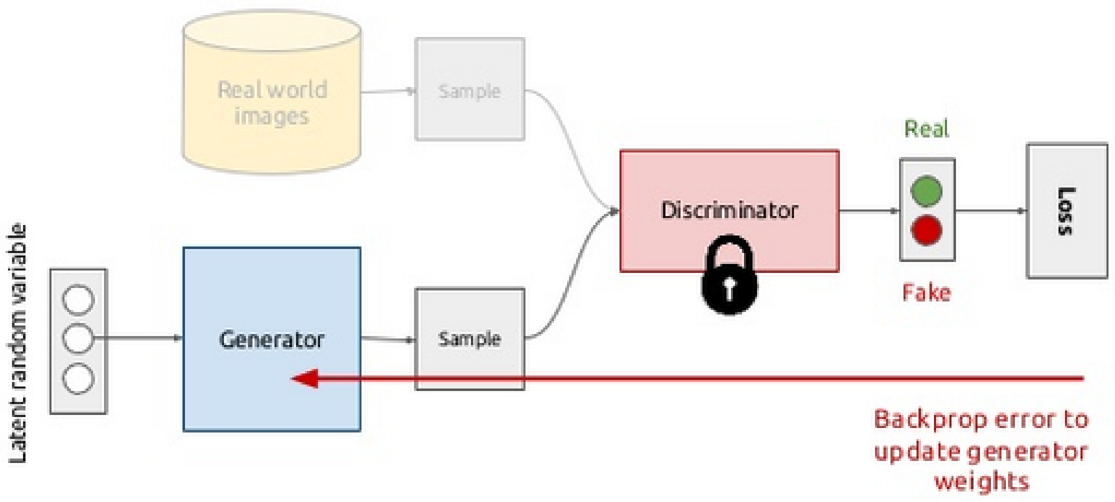


Figure 2.9: Generator Training Process [50]

The objective functions are learned jointly by alternating gradient descent. When G's model parameters are fixed and the first iteration is done, the D is trained by maximizing the Jensen-Shannon Divergence to distinguish between real and fake data (Figure 2.8). In the next iteration, the weights of D are fixed and G is trained by minimizing the Jensen-Shannon Divergence to make the generated data as realistic as possible (Figure 2.9).This is why the training of a GAN is defined as a 'minmax' game, and optimization can be done through any algorithm such as gradient descent, until a pre-defined number of epochs, or for as long as desired. The number of training epochs are generally decided based on whether the quality of generated images is as expected.

Limitations of Vanilla GAN

Although Vanilla GANs have achieved state of the art results in many domains, they suffer from certain drawbacks which have been outlined in this section:

- ***Nash Equilibrium is Hard to Achieve:*** The training process of GAN is based on gradient descent. The two models, generator and discriminator, are simultaneously trained to find a Nash Equilibrium. However, since both models update their loss functions concurrently and independently, there is no guarantee of convergence [52]
- ***Vanishing Gradient Problem:*** Training a GAN loss function poses a dilemma. If the discriminator is trained perfectly (especially early on in the training process), then ***$D(x_{real})=1$*** and **$D(x_{real})=0$**. From Equation (8) it can be observed that in this case, the value of the loss function would become 0, and there would be no gradient left to

update during the training iterations, hence leading to the vanishing gradient problem. However, if the discriminator function is not trained to perfection then the generator would not receive relevant feedback, meaning that the learned loss function would not be good enough to generate realistic images

- ***Mode Collapse:*** This is a common failure observed in GANs where the generator achieves a state where it always produces the same images as outputs. This may in some cases be enough to fool the discriminator, but the low variety of images generated is not representative of the complexities observed in real-world data distribution [53]
- ***Lack of Proper Evaluation Metrics:*** There is no numerical value to signify training progress. Therefore, the only way to tell when to stop the training process is by inspecting the samples being generated, which is a highly subjective method

As a result of these issues, many extensions and alternatives of the GAN model have been proposed, like Deep Convolutional GAN [54], Conditional GAN [55], Least Square GAN [56], etc., but most of these use a more or less similar objective function as the original GAN, and despite improvement in performance, still suffer from some of the same drawbacks.

Therefore, in the next sections, we have discussed the GAN variations which employ significantly different objective functions the one used in Vanilla GAN.

2.3.2 Wasserstein GAN (WGAN)

The WGAN architecture [57] replaced the Jensen-Shannon divergence from the original GAN architecture [12] with the Wasserstein Distance function[7] [58] ***W(P,Q)***, which is a measure of distance between the points in probability distributions ***P*** and ***Q***.

Using Wasserstein distance and following the same naming conventions as in Section 2.3.1, the value function for WGAN can be represented as Equation (9).

$$min_{\mathrm{G}} max_{\mathrm{D}\,\in\,\delta}(V, D) = E_{\mathrm{x}\,\in\,\mathrm{p_{real}}}[D(x)] - E_{\mathrm{z}\,\in\,\mathrm{p_z(z)}}[D(G(z))] \tag{9}$$

In Equation (9), δ denotes the set of 1-Lipschitz functions[8], meaning that the discriminator loss should follow the Lipschitz constraint.

[7]Wasserstein Distance is also called the "Earth Mover Distance", as it can be informally interpreted as the minimal cost of moving and transforming some quantity of mass (say, a pile of dirt) from the shape of one probability distribution ***P*** to that of another probability distribution ***Q***. The cost of moving in this scenario is calculated as the product of the amount of mass moved and the distance by which it has been moved.

[8]A Lipschitz function is a function ***f*** such that ***|f(x)-f(y)|*** $\leq$ ***K|x-y|*** for all ***x*** and ***y***, where ***K*** is a constant independent of ***x*** and ***y***.

The training process of WGAN is similar to that of Vanilla GAN, except that the discriminator is no longer directly learns to tell fake samples from real ones. Instead, it is trained to learn the K-Lipschitz continuity function to help in computing the Wasserstein distance. Since the discriminator in this architecture is trained for regression rather than classification, the sigmoid activation function is also removed, and the log function is no longer adopted in the loss function. During the training process as the value of loss function decreases, the Wasserstein distance also gets smaller, bringing the distribution closer and closer to that of the real data. However, since it is vital and tricky to maintain the continuity of this function, weight-clipping is introduced. After every gradient update on the discriminator loss function, the weights in each layer are clamped to the range of [-*c*, *c*].

WGAN Advantages over Vanilla GAN

WGAN overcomes the limitations of WGAN as follows:

- Since in this architecture, the discriminator is trained to minimize the Wasserstein distance metric instead of acting as a direct classifier between real and fake images, achieving convergence is much simpler than in Vanilla GAN
- The Wasserstein distance function is continuous, which makes the overall training process much more stable (as compared to Jensen-Shannon divergence used in Vanilla GAN, which is discrete in nature)
- The value function of WGAN would also reflect the quality of the generated sample. Lower values of loss (Wasserstein distance) would result in higher quality images, thus providing a quantifiable metric to measure the training progress
- The issues of vanishing gradient as well as mode collapse are resolved, thus ensuring diversity of training samples

Limitations of WGAN

Although the WGAN architecture addresses most problems, it introduces some of its own due to the use of weight-clipping to satisfy the Lipschitz constraint. Some of them are [59]:

- ***Vanishing and Exploding Gradients:*** When the clipping window is too small, it would constrict the weights of each layer in the discriminator to a very small range, which may lead the Gradient to either grow or decay exponentially, leading to vanishing gradient or exploding gradient problems

- ***Slow Convergence:*** When the clipping window is too large, it would lead to a very slow decrease in the Wasserstein distance, hence leading to slow convergence
- ***Not suitable for Very Complex Data:*** Weight clipping causes the discriminator to be biased towards simpler functions, i.e., to model very simple approximations to the value function. This is an underuse of the capacity of WGAN and may not be enough to fit complex data samples

To improve this architecture, weight-clipping is replaced in [59]. This is detailed in the next section.

2.3.3 Wasserstein GAN with Gradient Penalty (WGAN-GP)

In [59], imposing a Gradient Penalty instead of weight clipping as a means to enforce Lipschitz constraint is presented. The value function for WGAN-GP can be observed in Equation (10). Here, ***L*** represents the loss function, x' represents a sample from fake or generated data, and represents randomly sampled data. Note that a "soft penalty" is imposed (i.e. only on the randomly sampled data) to prevent tractability issues. The last term in the equation is the penalty term, with λ being the penalty coefficient.

$$L = E_{\mathrm{x} \,\in\, \mathrm{P_{real}}}[D(x)] - E_{\mathrm{x'} \,\in\, \mathrm{P_{fake}}}[D(x')] - \ \lambda \ \mathrm{E}_{\hat{x} \,\in\, \mathrm{P}_{\hat{x}}} \ [(\|\nabla_{\hat{x}} \ \mathrm{D}(\hat{x})\|_2 - 1)^2](10)$$

Generally, for a 1-Lipschitz function, the maximum gradient norm should be 1. Therefore, instead of applying weight-clipping, WGAN-GP loss function imposes a penalty if the gradient norm moves away from its maximum target norm value of 1.

Advantages of WGAN-GP

WGAN-GP overcomes the limitations of WGAN and presents a much more efficient architecture. Some key advantages over both Vanilla GAN and WGAN are listed below [60]:

- **Fast Convergence:** Since the penalty term in the loss function forces all gradient norms to go towards 1 when the discriminator is optimized, a unit gradient norm is observed in the real and fake data distributions. This makes the WGAN-GP converge much faster
- **Stable Training:** The training and optimizing process of WGAN-GP is much more stable even with untuned default parameters and not carefully designed architecture

- **Quality Output:** The meaningful loss function of this architecture helps determine when the training process should be stopped
- ***High Quality Output:*** WGAN-GP generates samples of much higher variety, with the least noise as compared to the other two architectures, which has also been verified by [61]

For the above mentioned reasons, WGAN-GP is a suitable model for a wider range of applications and datasets.

3 Design

This chapter introduces the high-level design of the various architectures used in this study, including the classification architecture, geometric transformation architecture (for data augmentation) and the WGAN-GP architecture, along with the relevant qualitative and quantitative metrics used to evaluate their performance. We have also summarized the overall workflow of this study to ensure the clarity of the related outlined experiments in Chapter 4.

3.1 Oversampling using Geometric Transformations

Some of the most common geometric and photo-metric transformations[1] applied on images to generate additional data while preserving the context of the image are rotation, translation, shearing, scaling, flipping, zooming, blurring, whitening etc. In order to apply these operations on minority classes, we will use the ImageDataGenerator class of the image pre-processing module of Keras [62].

ImageDataGenerator allows us to [63]:

- Generate batches of tensor image data with real-time augmentation, which is looped over (in batches)
- Configure randomized data transformations and/or normalization operations, which are done on the training data
- Create data generator instances for batches of augmented images (with labels)

A general workflow of data augmentation using ImageDataGenerator can be observed in Figure 3.1 above. Using this type of data augmentation we can ensure that the network would see new variations of the existing data at each and every epoch during training.

[1]For more details on the mathematics behind some of these 2D image transforms refer here: https://www.tutorialspoint.com/computer_graphics/2d_transformation

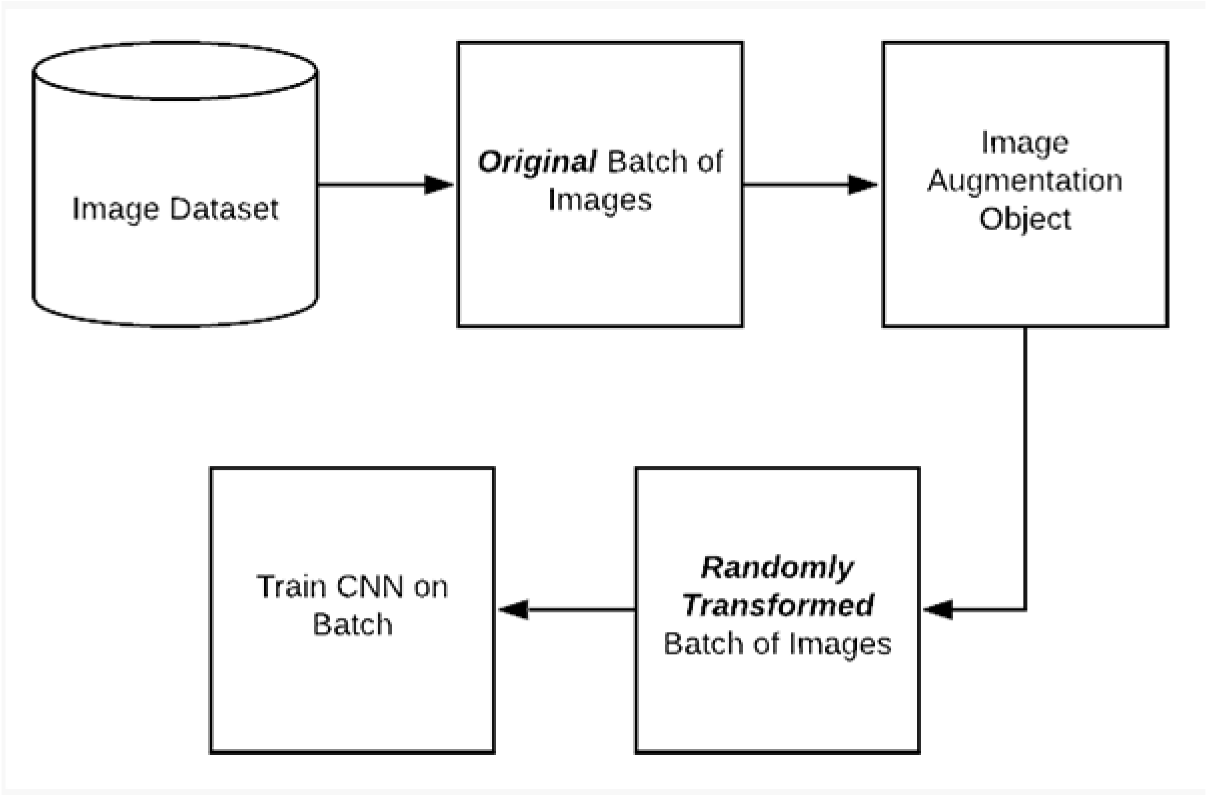


Image Dataset
Original Batch of Images
Image Augmentation Object
Randomly Transformed Batch of Images
Train CNN on Batch

Figure 3.1: Data Augmentation Approach of ImageDataGenerator [64]

3.2 Architecture of WGAN-GP

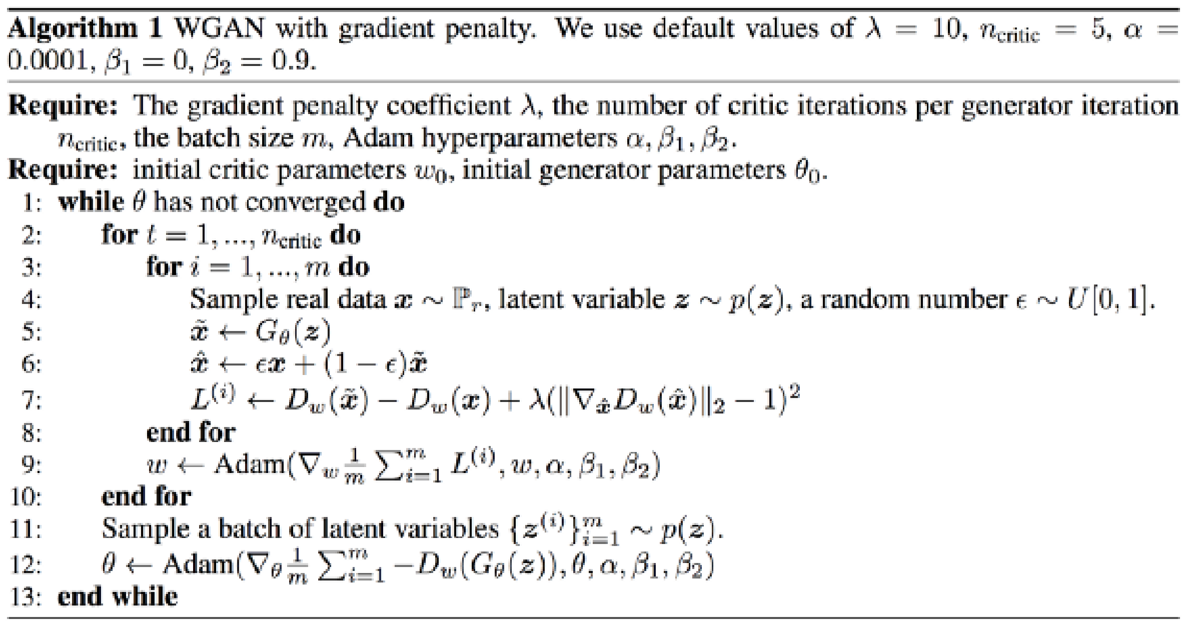
As highlighted in Chapter 2, WGAN-GP is a superior GAN architecture which is known to achieve convergence without facing the issues of vanishing or exploding gradients. The training process is very stable for this architecture, and it has also proven to generate highly diverse data samples with low noise. It can achieve high quality results with almost no hyperparameter tuning, making it a suitable choice for a wide variety of applications and datasets. Therefore, we have adopted the WGAN-GP architecture for our study.

The basic architecture, training process and the specific loss function of WGAN-GP have been explained in Chapter 2. The training algorithm of WGAN-GP is detailed in Figure 3.2.

The general methods for GAN architecture construction and the choices made in our model are described in this section. Some of these are based on [54] and [65]:

- ***Use of Leaky ReLU:*** The Rectified Linear Unit (ReLU)[2] activation function is a

[2]ReLU is a linear activation function unlike others, making it easy to optimize. For more details on ReLU refer here: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/



Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.

```
1: while $\theta$ has not converged do
2:     for $t = 1, ..., n_{\text{critic}}$ do
3:         for $i = 1, ..., m$ do
4:             Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
5:             $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
6:             $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
7:             $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
8:         end for
9:         $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
10:    end for
11:    Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
12:    $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
13: end while
```

Figure 3.2: Training Algorithm of WGAN-GP [59]

widely adopted, efficient activation function that returns the input directly as the output, or returns 0 when the input is 0.0 or less. However, the best practice for GANs is to use a variation called LeakyReLU, which allows some values lesser than 0, and learns the optimum cut-off for each node. In our architecture, we have used LeakyRelu in both the generator as well as discriminator, with slope values being of the order of the default value, 0.2.

- ***Use of Batch Normalization:*** Batch normalization is a technique used to improve the speed, performance and stability of neural networks by normalizing the input layer by adjusting and scaling the activations [3]. We employ batch normalization after the convolution layers. In the case of GANs, it helps avoid vanishing and exploding gradients, as well as mode collapse.

- ***Using Gaussian Weight Initialization:*** Before starting the training process, the weights (parameters) of the neural network must be initialized with small random variables to prevent the activation layer from producing vanishing or exploding outputs, which would cause very small or very large gradient updates, giving rise to convergence problems. It is considered to be a good practice to initialize all weights using a zero-centred Gaussian distribution, with mean value as 0 and variance value as 1/N, where N specifies the number of input neurons. Therefore, we have used Xavier initialization [4] in our architecture, which is based on the same principle.

[3]For more details on Batch Normalization, refer here: https://machinelearningmastery.com/how-to-accelerate-learning-of-deep-neural-networks-with-batch-normalization/

[4]For more details on Xavier Initialization, refer here: https://prateekvjoshi.com/2016/03/29/understanding-xavier-initialization-in-deep-neural-networks/

- ***Not using Max Pooling:*** A max-pooling layer is often used in CNNs to after each convolution layer to downsample the input and feature maps. However, we would not be using this approach as in the case of Generative Adversarial Networks, it has been shown that having all convolutional layers allows the network to learn its own spatial down-sampling, which leads to an increase in performance.

3.2.1 Evaluation Metrics for WGAN-GP

The training process of WGAN-GP is much more stable than that of Vanilla GAN and can achieve convergence, but practically achieving convergence for the model in all cases is not possible [66]. Therefore, in image-related problems, one common way to determine when the generated samples become satisfying and realistic is by visual inspection. Therefore, we will adopt this strategy to determine the number of training epochs, and to identify which generated samples we can add to our dataset.

The 'Real or Fake?' Human Perception Test

For re-validating the fact that the images being generated are photo-realistic, an experiment can be conducted for each imbalanced class.

Ten of the fake images generated using WGAN-GP for the class in question are randomly mixed with ten images sampled for that class from the real dataset. Using their best judgement, a human subject is made to mark each of the twenty images as "real" or "fake", without knowing the ground truth. This subject should not be aware of the distribution of real and fake samples in the dataset, so that the labeling is not influenced by unrelated assumptions.

This experiment can be conducted on a larger scale by sampling a bigger proportion of both datasets (real and fake), and by involving multiple human subjects. In this study, I myself have acted as the subject for this experiment.

Qualitative Evaluation Metrics for WGAN-GP

Since our goal is to use the WGAN-GP model to generate additional minority class images in order to augment an imbalanced dataset and balance it, we aim to fulfil the following criterion for our generated images through visual inspection [67]:

a. Generated images should be similar to the other images of the class in question. If this target is not met, it would mean that the generator is not trained enough to produce quality, realistic images

b. Generated images must not all be the same, or repetitive. This will ensure that the generator does not suffer from mode collapse problem

c. Generated images should be different from the images which are already present in the training set. If not, it would mean that we have simply trained our generative model to repeat the training data

Quantitative Evaluation of WGAN-GP Results using SSIM

To evaluate the above mathematically, the **Structural Image Similarity (SSIM)** metric [68] is used. SSIM measures the human perceptual difference between two similar images. We assign a value of 1 to SSIM when the two images are identical and as the differences between the two images become more observable, the value is decreased (never below 0).

To verify the diversity of generated images, a number of images are repeatedly generated for each class, and their SSIM is calculated.

To assess if the generated samples are diverse as compared to the pre-existing training data, the SSIM between generated images and the closest available sample is computed.

3.3 Architecture of Classification Model

In this research, we have used a modified version of Convolutional Neural Network, which is a hybrid of CNN and SVM, as it has shown to perform better on object recognition tasks as compared to using only CNN [69].

This hybrid architecture of CNN and SVM is expected to outperform the individual architectures, as it incorporates the merits of both SVM and CNN architectures and reduces their limitations. Some such merits are listed below [70]:

- The learning method of CNN is theoretically the same as that of MLP, where Empirical Risk Minimization (minimizing errors in training set) is used. When backpropagation is used and the first decision boundary or hyperplane is found, the training process is stopped and no attempt is made to improve the solution, regardless of whether the separating hyperplane found is at the local or global minima. This reduces the generalization ability of MLP.

  On the other hand, in SVMs, Structural Risk Minimization is used and a fixed training distribution is considered. The separating hyperplane obtained in SVM is a global optimum solution, reached when the maximum distance from support vectors on either side is achieved. Therefore, the generalization ability of SVMs is maximized, hence enhancing the classification accuracy of the hybrid model.

- MLP tends to assign high confidence values to misclassified samples located near the decision boundary, due to which such errors are not recognized later on.

  However, the SVM calculates the estimated probability of each class on one test data, providing more reliable label predictions. This also helps to develop a more efficient mechanism for the rejection of wrong results.

- The CNN architecture is an automatic salient feature extractor. These features are invariant to shift and shape distortions due to the shared weights on feature map. On the other hand, SVMs would require a custom-designed feature extractor, which would not be strong enough to take into account the connections between nearby pixels in an image. Therefore, CNN is a far superior feature extractor than SVM.

- SVM suffers from $O(n^2)$ time complexity, which makes it unsuitable for large datasets [71]. A common approach adopted to work on SVMs with bigger datasets is to sample a portion of data from each class and train it on reduced number of samples. This leads to inefficient or insufficient information extraction. However, in our architecture, this would not pose a problem as the features are learned using the CNN, not SVM.

In this model, the trainable CNN architecture extracts features from the data, and the SVM is used to recognize unknown patterns. A simple CNN structure from [72] is adopted as a feature extractor and can be observed in Figure 3.3. The input layer is a normalized matrix of size $S_1 X S_1$. The second and third feature map layers having N_1 and N_2 feature maps respectively are trained to do feature extraction at different resolutions. Each neuron on a feature map connects to 25 neurons in the previous layers, defined by $5 X 5$ convolution filters. The strength of this connection is defined by the weights in the kernel. All the neurons in one feature map share the same kernel and connecting weights. The latter of the architecture consists of fully-connected Multi-Layer Perceptron layer with N_3 neurons in the hidden layer and 10 neurons (for 10 classes) in the output layer.

Since a large number of weights need to be learned, therefore, a large training dataset is also required. Therefore, this is an ideal architecture to study the effects of imbalance in dataset and how classification accuracy increases when the total number of training samples increase as an effect of balancing the dataset. The hybrid architecture of CNN and SVM (Figure 3.4) is created by replacing the last output layer in Figure 3.3 with a multi-class SVM classifier.

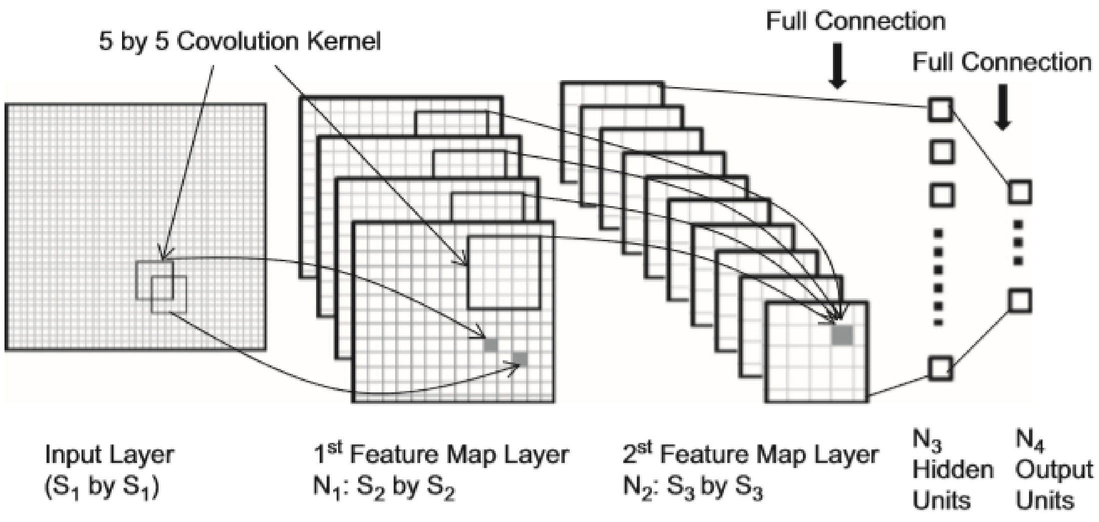


Figure 3.3: CNN Model used in Classification Architecture [69]

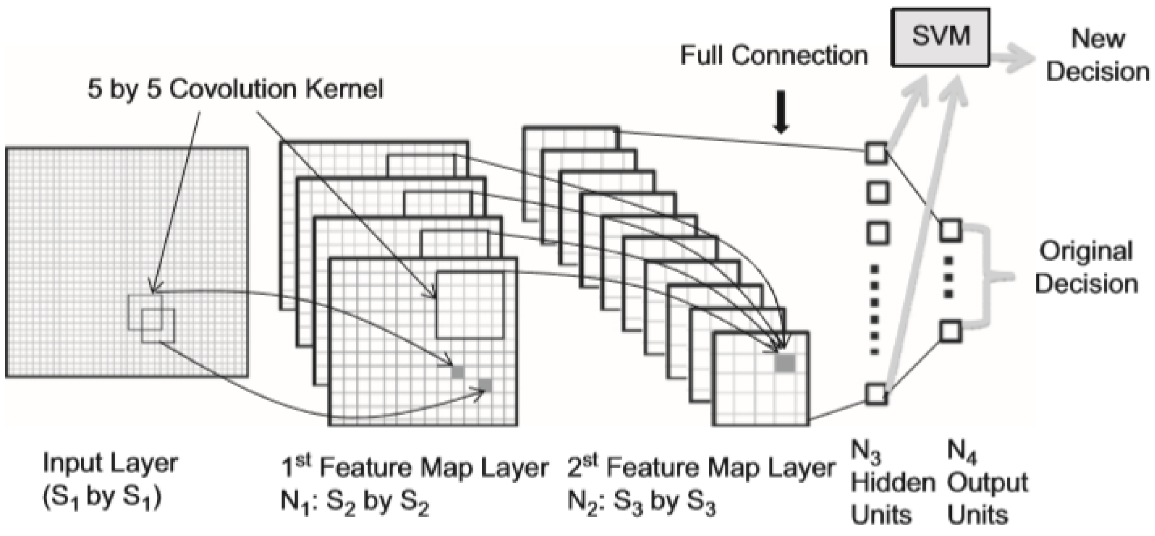


Figure 3.4: Hybrid Architecture of CNN and SVM [69]

The last layer of CNN outputs estimated probabilities for a given input which are calculated using an activation function which takes into account the learned weights and bias in the training process. However, this result cannot be comprehended by other classifiers. Therefore, the SVM uses this information as a feature vector in the hybrid architecture.

In the hybrid architecture, the original CNN (with the output layer) is trained on the input dataset until convergence is achieved. Then, the output layer is replaced with the Radial Bias Function (RBF) of SVM. The output from the CNN hidden layer is taken as a feature vector for training the SVM. Once trained, this SVM is able to perform the classification task on unseen data.

In particular, the CNN architecture adopted for FMNIST dataset (having 28 x 28 sized images) is shown in Figure 3.5

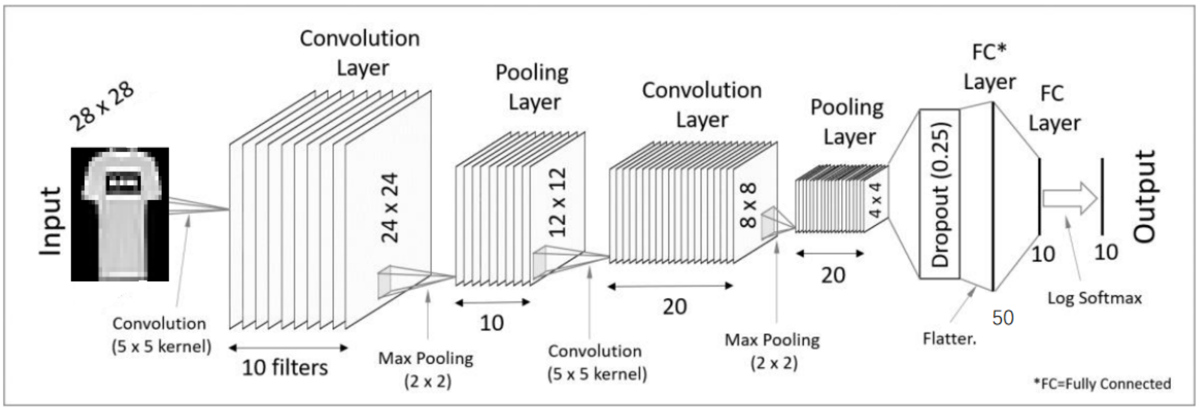


Figure 3.5: Architecture of the CNN model adopted for FMNIST (28 x28 x1 images)

The first convolutional kernel applied on the input images of 28x28x1 is of the order of 5x5. The ReLU activation function is used with it. Then, a 2x2 kernel is used to perform Max Pooling[5] with a stride value of 2 (so that the pooled regions remain non-overlapping). The second convolutional and pooling layers are similar to the first, but down-sampled, followed by a dropout regularization of 0.25. The dropout layer It randomly omits subsets of neurons (or features) at each iteration of the training procedure and helps overcome overfitting.

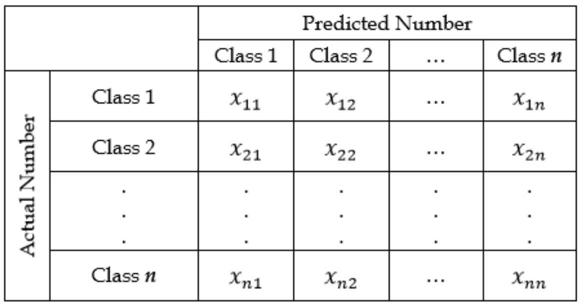
3.4 Performance Evaluation Metrics for Classification

To have a proper understanding of how a classification model is performing, it is important to evaluate it on multiple metrics.

These metrics are usually inferred from a **Confusion Matrix**, which can be seen as a summary of classification results (Predicted Numbers) on a set of testing data for which the actual values (Actual Numbers) are also known.

Consider a multi-class classification problem on a dataset containing n classes. The generalized format of the confusion matrix for the same can be observed in Table 3.1.

[5]Max-Pooling is a down-sampling technique to reduce the computational complexity and cost, and also as a measure to avoid overfitting. For more information on Max Pooling: https://computersciencewiki.org/index.php/Max-pooling/Pooling



		Predicted Number			
		Class 1	Class 2	…	Class n
Actual Number	Class 1	x_{11}	x_{12}	…	x_{1n}
	Class 2	x_{21}	x_{22}	…	x_{2n}
	⋮	⋮	⋮	⋮	⋮
	Class n	x_{n1}	x_{n2}	…	x_{nn}

Table 3.1: Confusion Matrix for Multi-Class Classification [73]

The minority classes are called the "positive classes" and the majority classes are called the "negative classes" In general, all the results obtained can be divided into four categories:

- **True Positives:** The cases in which the samples belonging to a positive class are correctly identified. The total number of True Positive instances in the system, across all classes *(TTP_{all})* can be calculated as Equation (1).

  In essence, it is the sum of the diagonal elements in the confusion matrix.

$$TTP_{all} = \sum_{j=1}^{n} x_{\mathrm{jj}} \tag{1}$$

- **False Negatives:** The cases in which data samples from positive classes get misclassified. The total number of cases of False Negatives for each class i ($\mathrm{TFN_i}$) can be calculated as Equation (2).

  In essence, it is the sum of all elements in a column, except for the element which belongs to the diagonal (or True Positive).

$$TFN_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} x_{\mathrm{ij}} \tag{2}$$

- **False Positives:** The cases in which a sample from a negative class gets misclassified under other labels. The total number of cases of False Positives for each class i ($\mathrm{TFP_i}$) can be calculated using Equation (3).

  In essence, it is the sum of all elements in a row, except for the element which belongs to the diagonal (or True Positive).

$$TFp_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} x_{\mathrm{ji}} \tag{3}$$

- **True Negatives:** The cases in which a data sample belonging to a negative class is correctly classified. The total number of cases of True Negatives for each class i ($\mathrm{TTN_i}$) can be calculated as Equation (4).

  In essence, it is the sum of all the values of the confusion matrix, excluding the row and column belonging to the class *i*.

$$TTN_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{\substack{k=1 \\ k \neq i}}^{n} x_{\mathrm{jk}} \tag{4}$$

3.4.1 Accuracy

Accuracy is the most widely used metric for evaluating classifier performance, and it can be computed using (5) for multi-class datasets. Accuracy represents the ratio of the number of correct predictions to the total number of predictions.

$$\text{Overall Accuracy} = \frac{\mathrm{TTP_{all}}}{\text{Total Number of Testing Entries}} \tag{5}$$

However, in case of imbalanced datasets, the Accuracy metric has a number of limitations [74].As an example, if the majority class represents 99% of all cases, there is a high probability of the classifier assigning the majority class label to all test cases, leading to classifier accuracy of 99%, which is misleading as in most cases the correct identification of the minority class samples are of more interest to us.

More specifically, if the train set is imbalanced and the test set is balanced, the decision threshold moves to reflect the estimated class prior probabilities and cause a low accuracy measure in the test set while the true discriminative power of the classifier does not change.

Therefore, we evaluate the classification performance on other metrics as well.

3.4.2 Precision

Precision measures how many predictions were correct, i.e., the ratio of how many of the predicted labels are actually present in the ground truth of the dataset. For multi-class datasets, the Precision for each class *i (P_i)* can be calculated using (6).

$$\mathrm{P_i} = \frac{\mathrm{TTP_{all}}}{\mathrm{TTP_{all}} + \mathrm{TFP_i}} \tag{6}$$

Since precision takes into account the number of incorrectly labelled samples, it is a metric that is sensitive to class imbalance.

However, by itself, this metric is insufficient as it provides no insight to the number of samples from the positive group which were mislabelled as negative. Note that in case of imbalanced datasets, majority classes are usually called "negative" and the minority classes are called "positive".

Therefore, Precision needs to be studied in conjunction with another metric called Recall.

3.4.3 Recall

Recall, or True Positive Rate is a ratio of how many predicted labels are the same as actual labels to the total number of actual labels or data samples for a class i. For multi-class datasets, the Recall value for each class ($\mathrm{R_i}$) can be calculated using (7).

$$\mathrm{R_i} = \frac{\mathrm{TTP_{all}}}{\mathrm{TTP_{all}} + \mathrm{TFN_i}} \tag{7}$$

Since Recall is only dependent on the positive group, it is not affected by imbalance.

However, it does not consider the number of negative samples that are misclassified as positive, which may be problematic for imbalanced datasets with many negative samples.

Note that in case of imbalanced datasets, majority classes are usually called "negative" and the minority classes are called "positive".

Therefore, there is a trade-off between precision and recall, and which metric should be given more weightage is problem-specific.

3.4.4 F1-Score

The F-Measure of F1-Score ((8)) is a combination of precision and recall using harmonic mean, where coefficient β is used to adjust the relative weightage of precision to recall.

$$\text{F1-Score}_\text{i} = \frac{(1+\beta^2) \text{ * Recall * Precision}}{\beta^2 \text{ * Recall *Precision}} \tag{8}$$

When $\beta < 1$, more weight is assigned to precision (when $\beta \to 0$, only precision is considered) and when $\beta >1$, recall is favoured (when $\beta \to \infty$, only recall is considered).

In our case, since we are trying to obtain results for general scenarios and are considering multiple datasets, we assign β=1, hence giving equal weightage to both parameters.

The range of F1-Score is [0,1], where 1 is a perfect classification, i.e., perfect precision and recall values, and the worst is at 0.

3.5 Deep Learning Frameworks

Many deep learning frameworks are available as libraries for implementing machine learning models and neural networks, such as Tensorflow [75], Keras [14] and Pytorch [76]. This section briefly describes the structure and working of the frameworks employed in our implementation. [77].

3.5.1 Tensorflow

Tensorflow is an open source machine learning library, offering high flexibility and performance as well as portability among platforms and devices. It operates by developing a static computational graph for the operations, wherein nodes represent the functions or operators and the edges are data (tensors). The programming stack for tensorflow can be observed in Figure 3.6.It is great for back-end implementation and has a huge community support, making it a popularly adopted library.

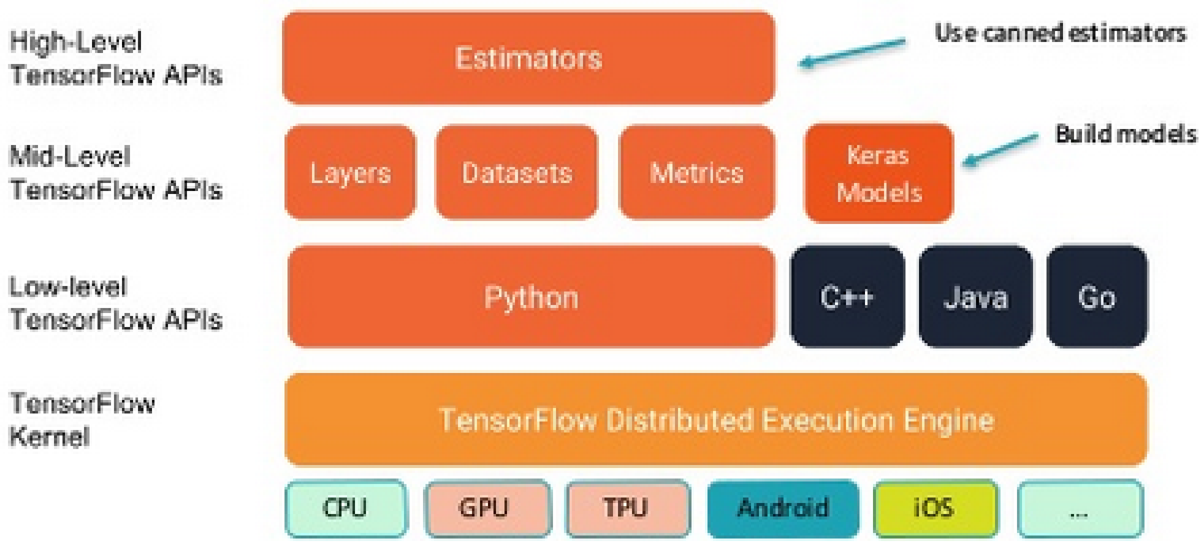


Figure 3.6: Tensorflow Programming Stack [78]

3.5.2 Keras

Keras is a front-end layer, which runs on top of other popular deep learning frameworks such as Tensorflow, Theano [79], Microsoft Cognitive Toolkit [80], etc. The Keras stack can be observed in Figure 3.7. It is a user-friendly API with built-in support and pre-defined functions for CNNs. It enables faster experimentation with neural networks as it supports aribitrary network architectures.

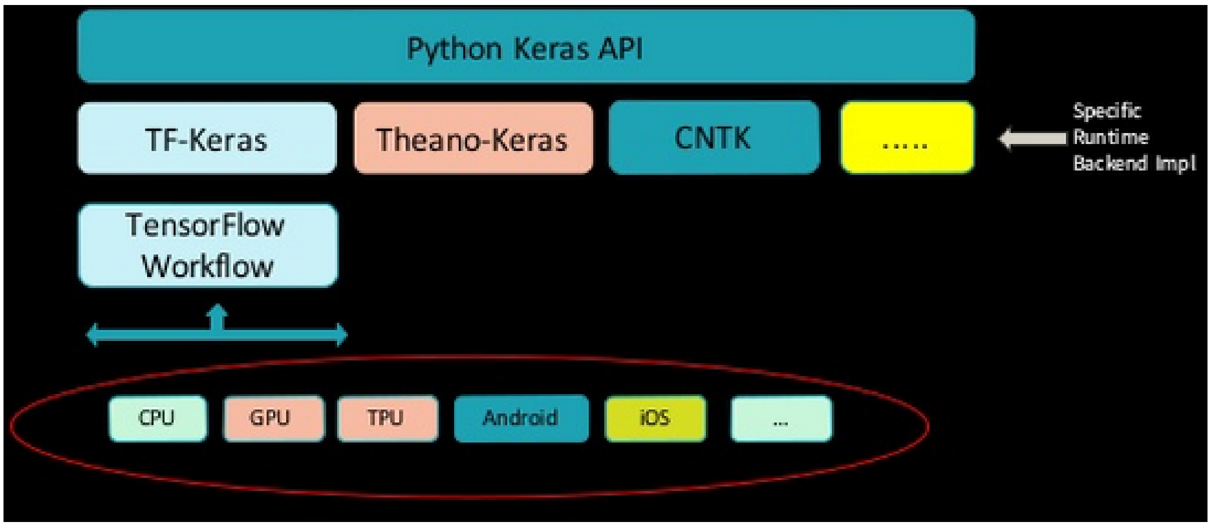


Figure 3.7: Keras Programming Stack [78]

3.6 Overall Workflow

The different modules and the overall workflow of the project are summarized in Figure 3.8.

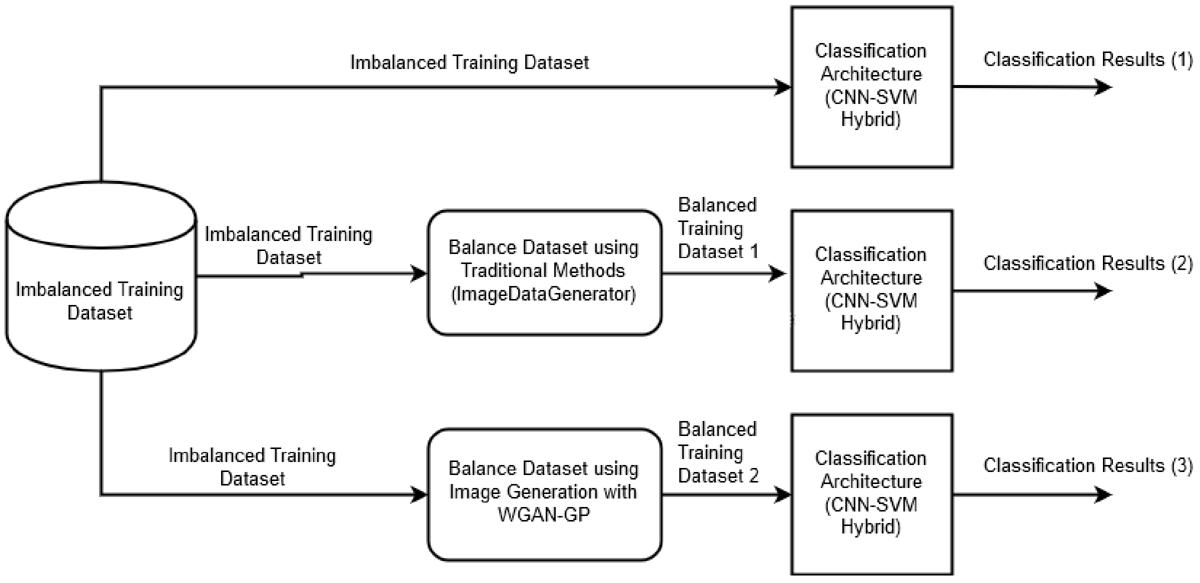

Imbalanced Training Dataset
Classification Architecture (CNN-SVM Hybrid)
Classification Results (1)
Imbalanced Training Dataset
Imbalanced Training Dataset
Balance Dataset using Traditional Methods (ImageDataGenerator)
Balanced Training Dataset 1
Classification Architecture (CNN-SVM Hybrid)
Classification Results (2)
Imbalanced Training Dataset
Balance Dataset using Image Generation with WGAN-GP
Balanced Training Dataset 2
Classification Architecture (CNN-SVM Hybrid)
Classification Results (3)

Figure 3.8: Workflow of the Dissertation

We will be first training the hybrid CNN-SVM classifier on the imbalanced datasets, and obtain the test accuracy, precision, recall and F1-score for the same. Then, we will perform dataset augmentation using traditional methods (geometric transforms and photo-metric transforms) to obtain a balanced dataset (Balanced Training Dataset 1), which would be used to re-train the classifier and to obtain the classification results. Finally, the dataset balancing process will be done using the WGAN-GP architecture, which would result in a balanced dataset again (Balanced Training Dataset 2), which would again be used to re-train the classifier and obtain results. The Classification Results (1), (2) and (3) are obtained on the corresponding test dataset, which remains unchanged from the original datasets. These results would then be compared and contrasted in Chapter 5 to gain a thorough understanding of the effect of data imbalance and data augmentation using different methods.

4 Implementation

This chapter makes use of the design decisions outlined in Chapter 3 to implement the frameworks required for conducting experiments relevant to the research question. The technical setup and the datasets used in this study are first explained, followed by the exact classifier model details and data augmentation using both traditional methods as well as using WGAN-GP.

4.1 Technical Setup

4.1.1 Coding-Related Details

The implementation of all modules and architectures used in this study has been done in Python 3, with Keras and Tensorflow as the deep-learning frameworks. Some other Python libraries used in the experiments are: scikit-learn, matplotlib, pandas and numpy.

4.1.2 Hardware Support

For general implementation, Google Colaboratory [81] has been used, which is a free Jupyter notebook environment, optimized for machine learning tasks with GPU and TPU support, platform-independent and having most of the dependencies resolved. The implementation was done on on 12 GB RAM with GPU support.

However, for the training of WGAN-GP, more powerful and customized hardware support is required. Therefore, we used Intel i7 7700k CPU, with one GTX 1080 GPU, on Ubuntu 16.04 Platform.

4.2 Datasets

For testing and evaluating our proposed methodology, we have considered two benchmark datasets: Fashion MNIST or FMNIST [10] and CIFAR-10 [11]. A brief summary of the two can be seen in Table 4.1. Note that these datasets have equal number of instances in each class and no natural imbalance can be observed in either of them.

S.No.	Dataset Name	Image Dimensions			No. of Classes	Total Images	
		Width	Height	Depth		Training Set	Test Set
1	FMNIST	28	28	1	10	60,000	10,000
2	CIFAR-10	32	32	3	10	50,000	10,000

Table 4.1: Dataset Summaries for Original Datasets [10, 11]

The characteristics and significance of these datasets have been briefly described in this section.

4.2.1 FMNIST

The Fashion-MNIST dataset contains 28 x 28 grayscale images of clothing articles. It has 60,000 images in the training set and 10,000 images in the testing set. The data is divided into ten, labelled from 0 to 9, which can be observed in Figure 4.1.

This dataset was created by Zalando Research [82] to "serve as a direct drop-in replacement of the original MNIST dataset for benchmarking machine learning algorithms" [83]. The MNIST dataset [46] is a benchmark dataset which is often the first point of validation for any machine learning or data science algorithm. However, a few reasons for not using it as pointed out in [83, 84] are:

- MNIST is a very easy dataset in which most pairs of data can be distinguished from one another by a single pixel only. This is not representative of real world datasets which are often much varied and complex, and therefore is not an appropriate benchmark for modern-day computer vision applications
- MNIST is an overused dataset, which, due to its simplicity, often wrongly proves inefficient algorithms applied on it to be efficient

Therefore, we have chosen FMNIST dataset as a benchmark to validate our research.

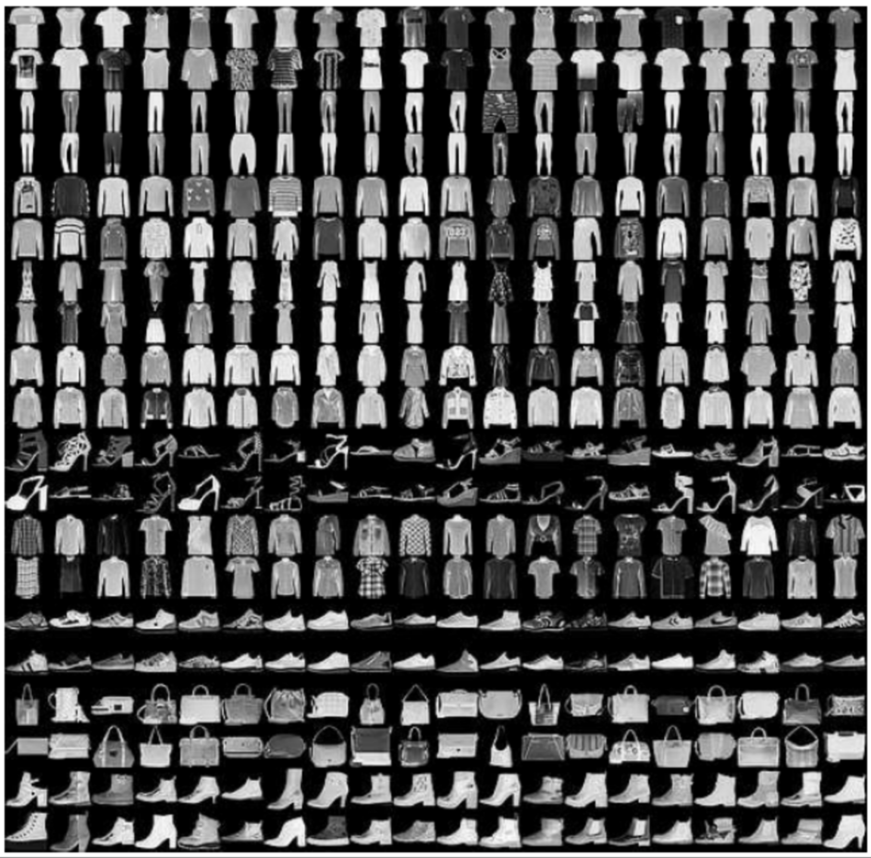
Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Figure 4.1: FMNIST Dataset Labels and Example Images [10]

However, as mentioned before, this dataset is perfectly balanced and each class contains exactly 6,000 train instances and 1,000 test instances. Therefore, for the purpose of studying the effects of class imbalance, we have dropped 5300, 5500, 5000 and 5450 instances from the training sets of the four classes of Dress (3), Coat (4), Sandals (5) and Sneaker (7) respectively. The imbalance created in the train set can be observed in Figure 4.2, where the X-axis displays the training labels (0 to 9) and the Y-axis depicts the corresponding frequency of data samples.

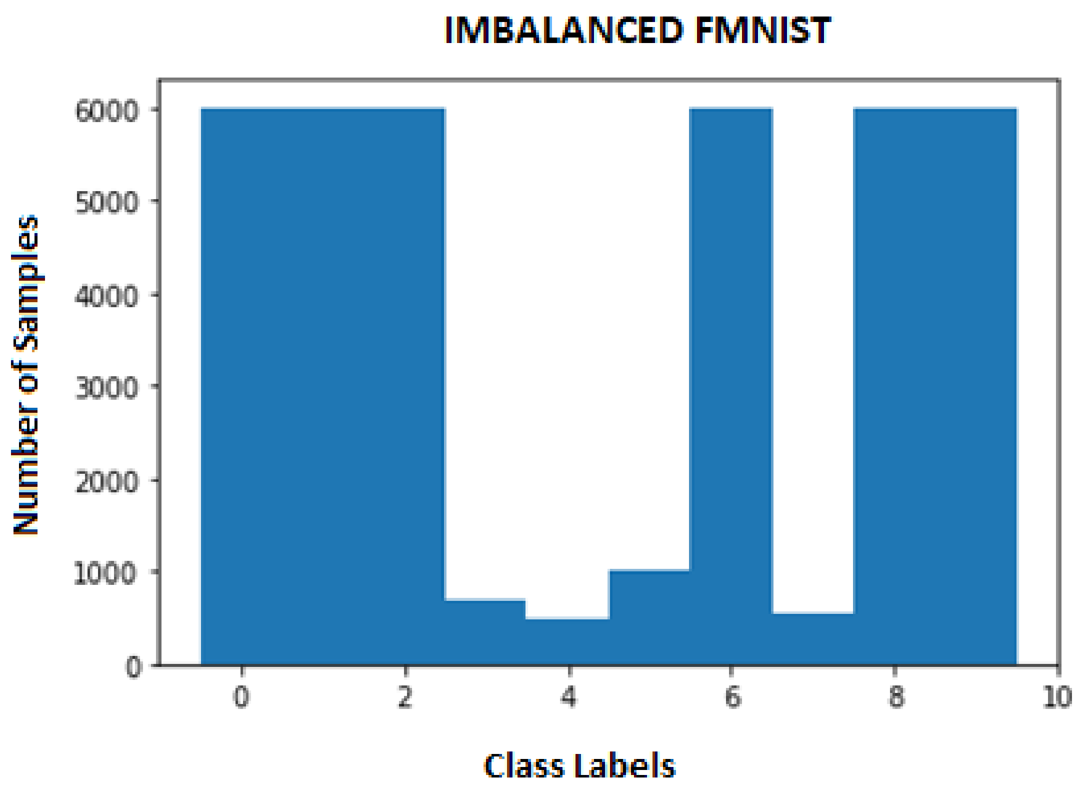


Figure 4.2: Class Distribution in FMNIST Dataset After Imbalancing

However, the test set is left untouched, and still contains 1000 instances per class.

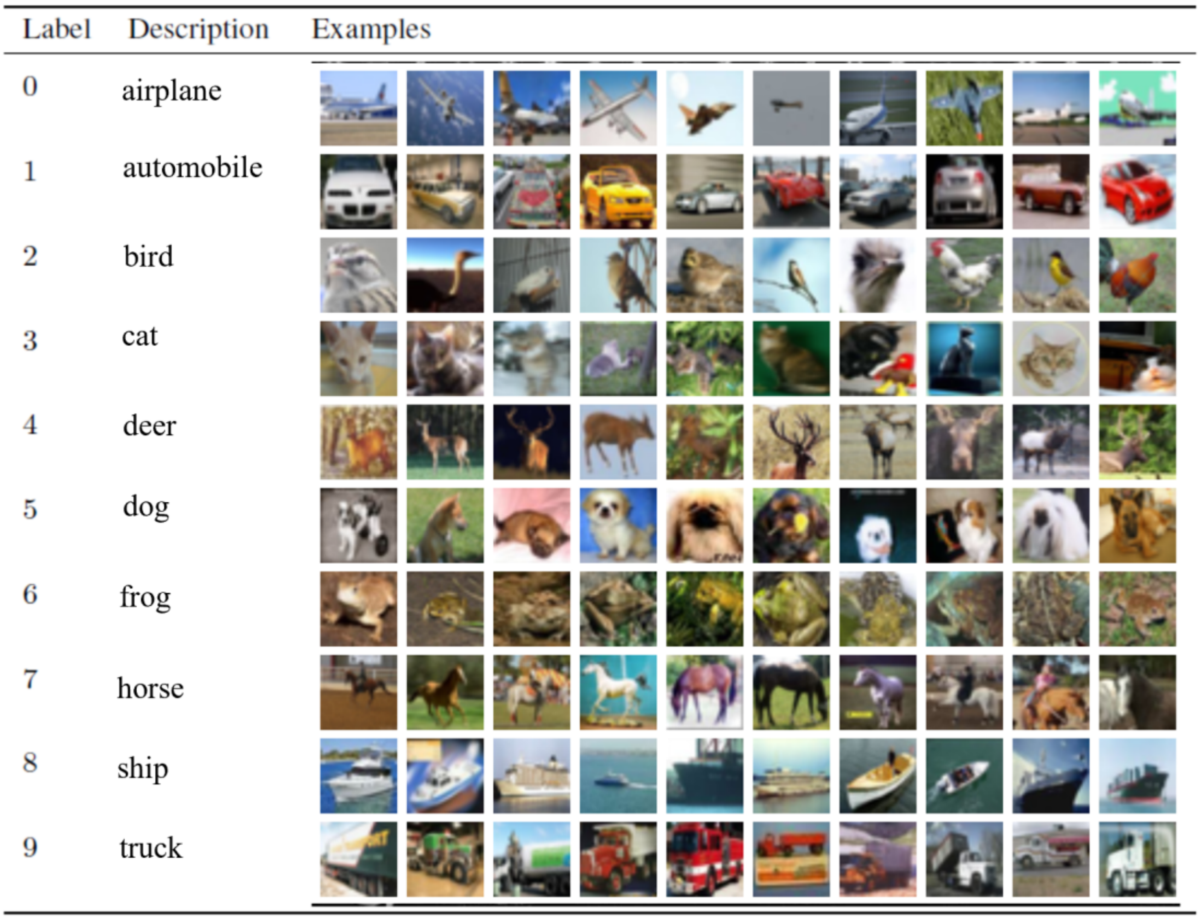
The imbalance ratio (Equation (1)) for this dataset would be ρ*=6000/500* or ρ*=12*.

4.2.2 CIFAR-10

The CIFAR-10 Dataset [11] is a subset of the Visual Dictionary or 80 Million Tiny Images Dataset [85], which is a comprehensive dataset of images gathered from the web.

It consists of 32x32 colour images divided into train and test sets of sizes 50,000 and 10,000 respectively. There are 10 mutually exclusive, non-overlapping classes, labelled internally from 0 to 9, and having more meaningful label names such as ‘airplane’, ‘bird’, ‘cat’, etc., as can be observed in Figure 4.3.

CIFAR-10 is one of the most widely used datasets in Machine Learning [86] and is more representative of real world problems than FMNIST as it contains coloured images of real objects. Therefore, the performance of our methodology on this dataset would be indicative of how it would perform on most image datasets pertaining to real-life applications.



Label	Description	Examples
0	airplane	
1	automobile	
2	bird	
3	cat	
4	deer	
5	dog	
6	frog	
7	horse	
8	ship	
9	truck	

Figure 4.3: Class Labels and Examples of CIFAR-10 Database

As with FMNIST, CIFAR-10 is naturally balanced, therefore, we force imbalance it by dropping 2500, 3500, 4000 and 4500 instances from the train set of classes automobile (1), bird (2), cat (3) and horse (7) respectively. The frequency of classes after introducing this imbalance can be observed in Figure 4.4. The X-Axis shows the labels of the training set (0 to 9) and the Y-Axis displays the corresponding number of data instances. Note that the test dataset is remained untouched.

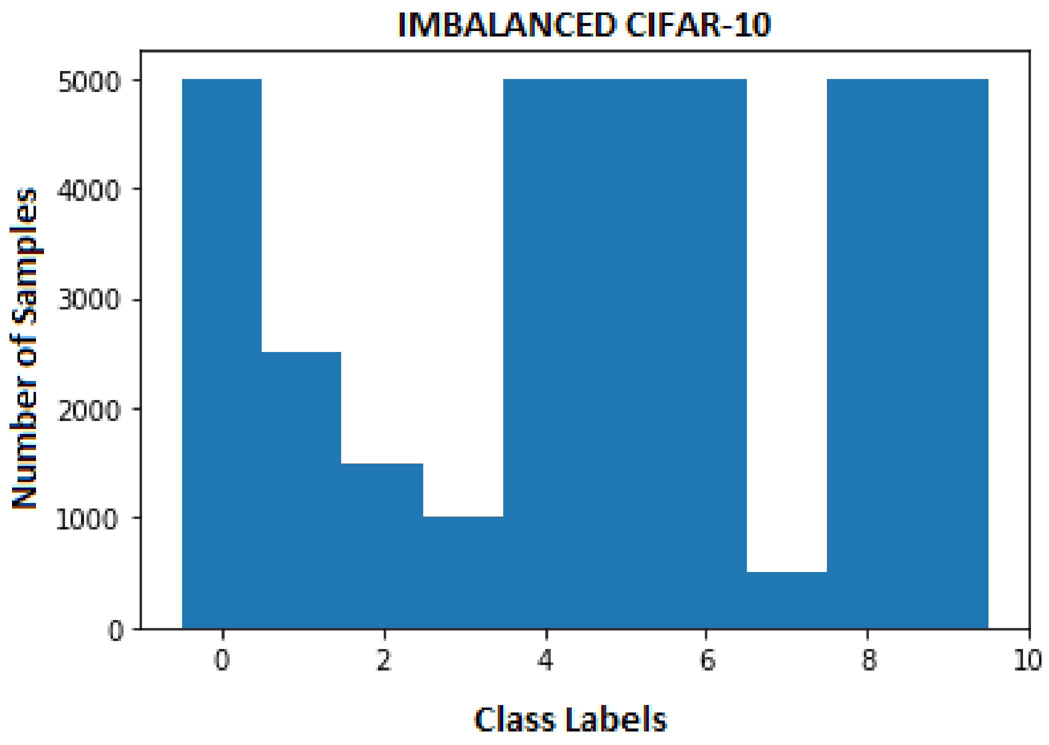


Figure 4.4: Data Distribution of CIFAR-10 after Imbalancing

Note that the number of instances dropped from each class is lesser than the images dropped from the classes of FMNIST. This is because CIFAR-10 is a more complex dataset, which means that not only the effect of imbalance would be observed more easily, but also that the training time would increase as the imbalance increases.

As before, the test set of the classes are not changed for the experiments.

The imbalance ratio (Equation (1)) for this dataset would be ρ=*5000/500* or ρ=*10*.

We will investigate the effects of imbalance on classification on both FMNIST and CIFAR-10 datasets in order to do a comparison of the results on datasets of increasing complexity and validate our research.

4.2.3 Dataset Augmentation by Image Transformations

In order to balance the imbalanced datasets, we generate additional data using ImageDataGenerator class of image pre-processing module of Keras (Refer Chapter 3), for the minority classes by applying various geometric and image transforms on the samples available to us (keras.preprocessing.image.ImageDataGenerator(arg1, arg2, ...)).[1]

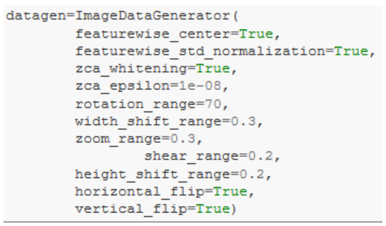
Some of the arguments of ImageDataGenerator [62] that we have used in our experiments are described below (Table 4.2):

[1]The code available here has been used as a reference: https://github.com/franneck94/MNIST-Data-Augmentation/blob/master/mnist.py

Argument	**Description**	**Data Type**	**Possible Values**
featurewise_centre	Set input mean to 0 over the dataset, feature-wise	Boolean	True, False
featurewise_std_normalization	Divide inputs by standard deviation of the dataset, feature-wise	Boolean	True, False
zca_epsilon	Epsilon value for ZCA whitening	Float	Any float value (Default: 1e-6)
zca_whitening	Apply ZCA whitening	Boolean	True, False
rotation_range	Degree range for random rotations	Int	Ideally, values between 0 to 360
width_shift_range	Range as a fraction of total width within which to randomly translate pictures vertically or horizontally	Float	Any float value. It is taken as a fraction of total width, if it is < 1, or as pixels if >= 1 Example: If width_shift_range=1.0 possible values are floats in the half-open interval [-1.0, +1.0[
		1D Array-Like	Random elements from the array. Example: width_shift_range=[-1, 0, +1]
		Int	Integer number of pixels from interval (-width_shift_range, +width_shift_range) Example: If width_shift_range=2 possible values are integers [-1, 0, +1]
height_shift_range	Range as a fraction of total height within which to randomly translate pictures vertically or horizontally	Float	Any float value. It is taken as a fraction of total height, if it is < 1, or as pixels if >= 1 Example: if with height_shift_range=1.0 possible values are floats in the half-open interval [-1.0, +1.0[.
		1D Array-Like	Random elements from the array. Example: height_shift_range=[-1, 0, +1]
		Int	Integer number of pixels from interval (-height_shift_range, +height_shift_range) Example: If height_shift_range=2 possible values are integers [-1, 0, +1]
shear_range	Shear Intensity (Shear angle in counter-clockwise direction in degrees)	Float	Any float value
zoom_range	Range for randomly zooming inside pictures	Float or [lower, upper]	Any float value or range If a float, [lower, upper] = [1-zoom_range, 1+zoom_range]
channel_shift_range	Range for random channel shifts	Float	Any float value
fill_mode	Strategy used for filling in newly created pixels, which can appear after a rotation or a width/height shift	String {"constant", "nearest", "reflect" or "wrap"}	"constant": kkkkkkkk|abcd|kkkkkkkk (cval=k) "nearest" (Default): aaaaaaaa|abcd|dddddddd "reflect": abcddcba|abcd|dcbaabcd "wrap": abcdabcd|abcd|abcdabcd
horizontal_flip	Randomly flip inputs horizontally	Boolean	True, False
vertical_flip	Randomly flip inputs vertically	Boolean	True, False

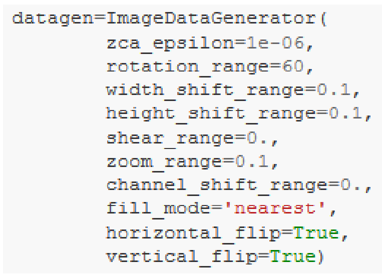
Table 4.2: Some ImageDataGeneration Arguments with Descriptions [62]

Specifically, the ImageDataGenerator arguments used for augmenting the training sets in imbalanced FMNIST and imbalanced CIFAR-10 datasets are shown in Figure 4.5 (a) and Figure 4.5 (b). The new images generated as a result of applying these can be observed in Chapter 5.



```
datagen=ImageDataGenerator(
        featurewise_center=True,
        featurewise_std_normalization=True,
        zca_whitening=True,
        zca_epsilon=1e-08,
        rotation_range=70,
        width_shift_range=0.3,
        zoom_range=0.3,
                shear_range=0.2,
        height_shift_range=0.2,
        horizontal_flip=True,
        vertical_flip=True)
```

(a) For Imbalanced-FMNIST



```
datagen=ImageDataGenerator(
        zca_epsilon=1e-06,
        rotation_range=60,
        width_shift_range=0.1,
        height_shift_range=0.1,
        shear_range=0.,
        zoom_range=0.1,
        channel_shift_range=0.,
        fill_mode='nearest',
        horizontal_flip=True,
        vertical_flip=True)
```

(b) For Imbalanced-CIFAR-10

Figure 4.5: Custom ImageDataGenerator Arguments used for the two datasets

Note that more randomness is introduced through the ImageDataGenerator arguments in augmenting the FMNIST dataset as compared to the augmentation for CIFAR-10, as FMNIST images are grayscale as well as simpler than those in the latter, and the possibility of generating same or similar samples, or not having enough variation in the generated images is higher. Variability of generated images is emphasized upon in our experiments to ensure that the model does not overfit, as that would affect the classification accuracy and results.

4.3 Dataset Augmentation using WGAN-GP

The WGAN-GP architecture[2] is developed using Tensorflow framework. The specific neural network models developed for the generator and the discriminator would vary for the two datasets due to the difference in image dimensions, and they have been summarized in the subsections below.

The training of WGAN-GP involves the simultaneous training of generator and discriminator, and as the number of training epochs increase, the image output becomes more and more realistic. For the purpose of this study, instead of waiting for model convergence to start sampling data for dataset augmentation, I monitored the outputs through each epoch, and sampled the data starting the epoch at which a realistic output was observed for the first time. Note that this observation is based on the personal perception of the researcher and can be influenced by their level of understanding of the datasets, coginitive ability, etc.

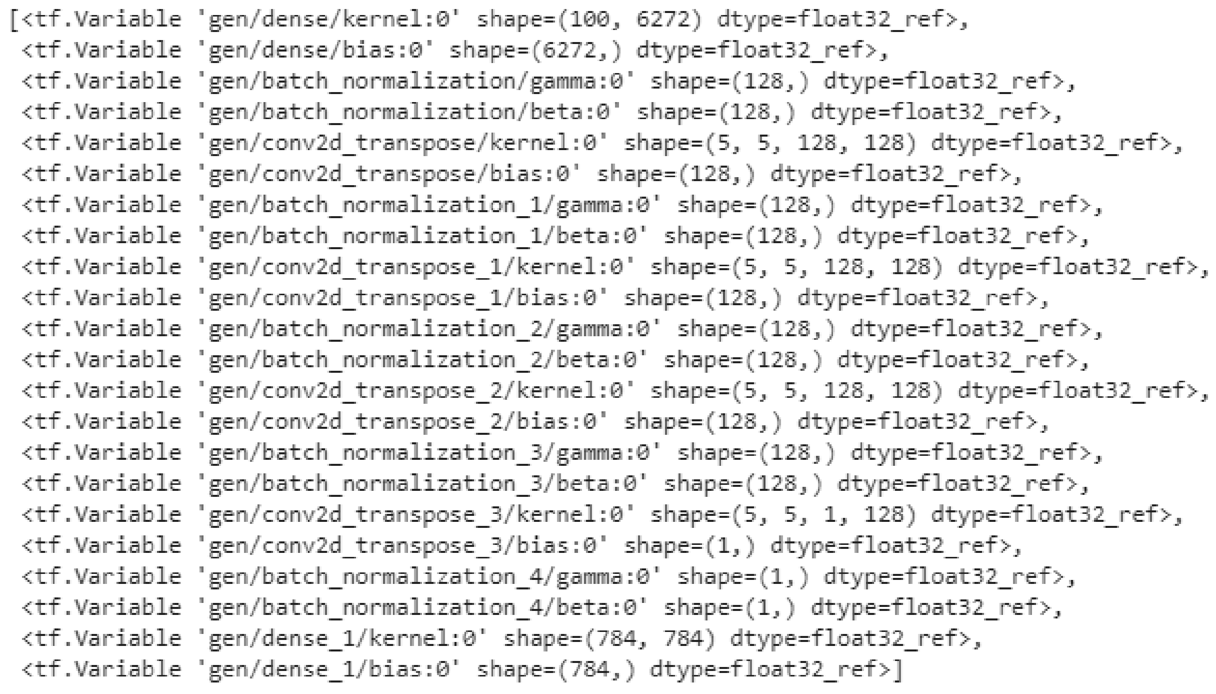
At the start of the training process, it can be observed that the generator's outputs are very

[2]The code available here has been used as a reference: https://github.com/caogang/wgan-gp

noisy, but start looking more and more realistic as the process goes on, as can be observed in Figures 4.7 and 4.9.

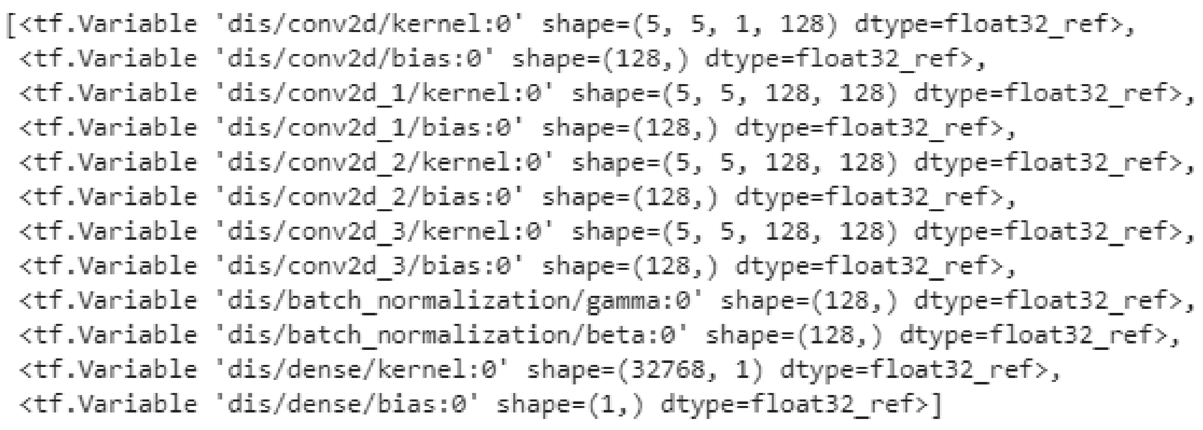
4.3.1 WGAN-GP Model Description for FMNIST

WGAN-GP was trained separately for each of the four minority classes of 'Dress', 'Coat', 'Sandal' and 'Sneaker' to generate images class-wise, until each class reached a total of 6,000 instances. The models of the generator and the discriminator of WGAN-GP for 28x28x1 images of FMNIST are described in Figures 4.6(a) and 4.6(b).



```
[<tf.Variable 'gen/dense/kernel:0' shape=(100, 6272) dtype=float32_ref>,
 <tf.Variable 'gen/dense/bias:0' shape=(6272,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization/gamma:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization/beta:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/conv2d_transpose/kernel:0' shape=(5, 5, 128, 128) dtype=float32_ref>,
 <tf.Variable 'gen/conv2d_transpose/bias:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization_1/gamma:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization_1/beta:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/conv2d_transpose_1/kernel:0' shape=(5, 5, 128, 128) dtype=float32_ref>,
 <tf.Variable 'gen/conv2d_transpose_1/bias:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization_2/gamma:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization_2/beta:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/conv2d_transpose_2/kernel:0' shape=(5, 5, 128, 128) dtype=float32_ref>,
 <tf.Variable 'gen/conv2d_transpose_2/bias:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization_3/gamma:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization_3/beta:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/conv2d_transpose_3/kernel:0' shape=(5, 5, 1, 128) dtype=float32_ref>,
 <tf.Variable 'gen/conv2d_transpose_3/bias:0' shape=(1,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization_4/gamma:0' shape=(1,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization_4/beta:0' shape=(1,) dtype=float32_ref>,
 <tf.Variable 'gen/dense_1/kernel:0' shape=(784, 784) dtype=float32_ref>,
 <tf.Variable 'gen/dense_1/bias:0' shape=(784,) dtype=float32_ref>]
```

(a) Generator Network

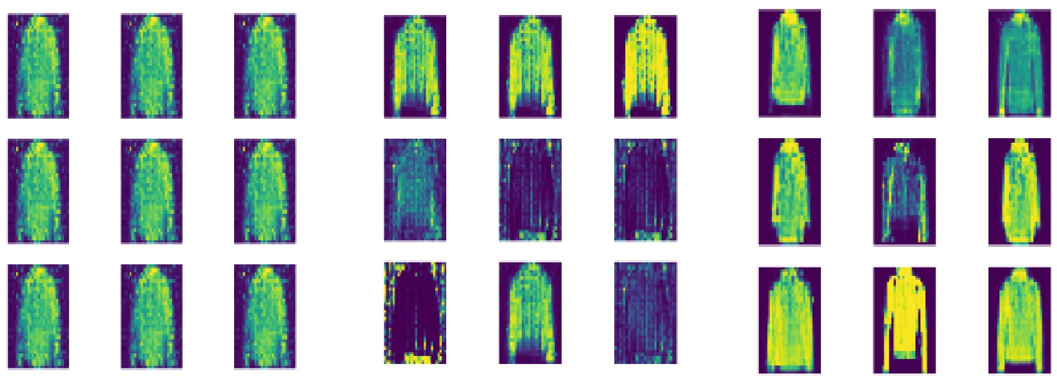


```
[<tf.Variable 'dis/conv2d/kernel:0' shape=(5, 5, 1, 128) dtype=float32_ref>,
 <tf.Variable 'dis/conv2d/bias:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'dis/conv2d_1/kernel:0' shape=(5, 5, 128, 128) dtype=float32_ref>,
 <tf.Variable 'dis/conv2d_1/bias:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'dis/conv2d_2/kernel:0' shape=(5, 5, 128, 128) dtype=float32_ref>,
 <tf.Variable 'dis/conv2d_2/bias:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'dis/conv2d_3/kernel:0' shape=(5, 5, 128, 128) dtype=float32_ref>,
 <tf.Variable 'dis/conv2d_3/bias:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'dis/batch_normalization/gamma:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'dis/batch_normalization/beta:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'dis/dense/kernel:0' shape=(32768, 1) dtype=float32_ref>,
 <tf.Variable 'dis/dense/bias:0' shape=(1,) dtype=float32_ref>]
```

(b) Discriminator Network

Figure 4.6: WGAN-GP Model Description for FMNIST Dataset

The results generated through some of the epochs can be observed in Figure 4.7. Notice that since this dataset contains smaller, less-complex images, training over a very few epochs can also give great results, and starting Epoch 25, realistic-looking samples are generated for class 'Coat'.

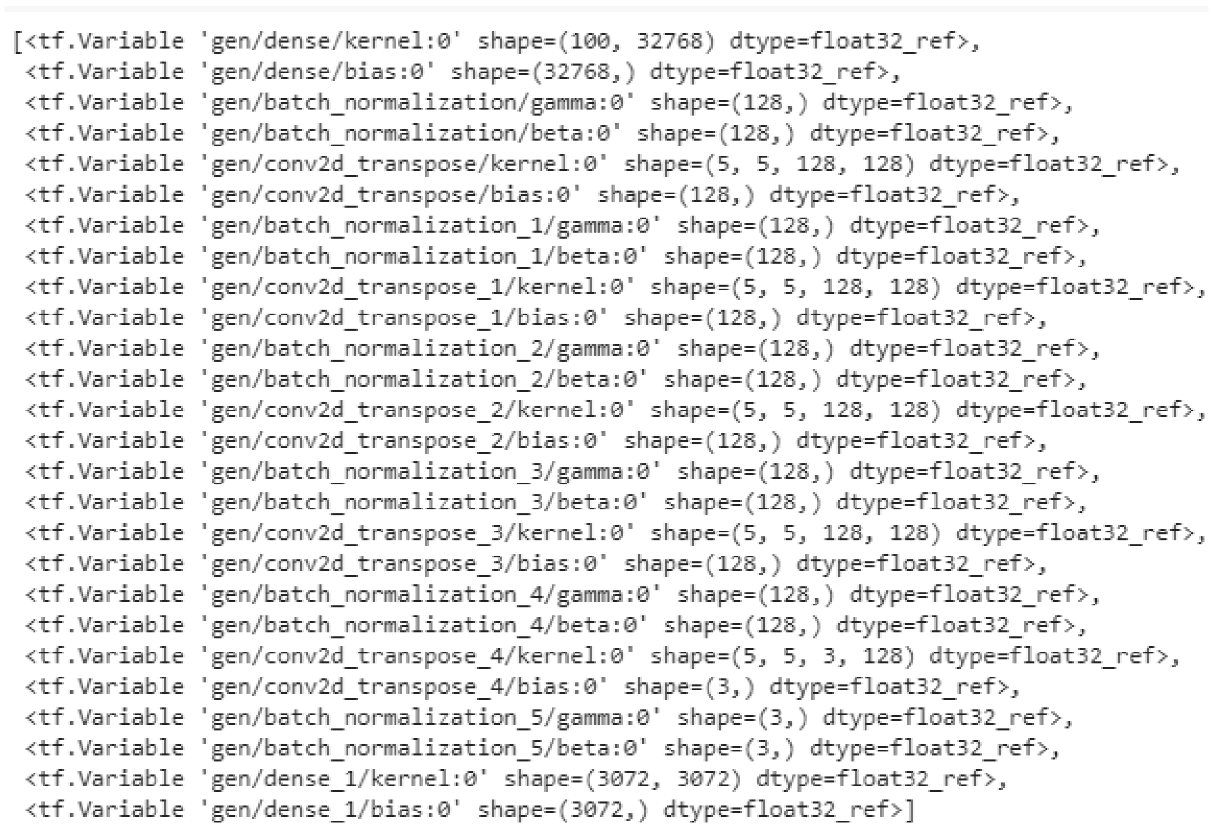


(a) Epoch 0 (b) Epoch 13 (c) Epoch 25

Figure 4.7: Data Generated for the Minority Class 'Coat' in Imbalanced-FMNIST using WGAN-GP

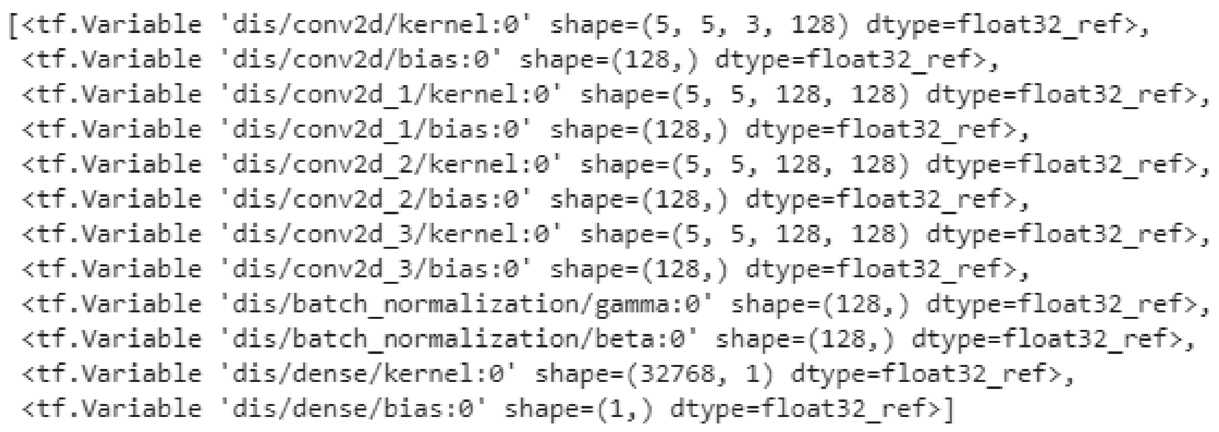
4.3.2 WGAN-GP Model Description for CIFAR-10

As in the previous section, WGAN-GP model for CIFAR-10 dataset was trained on the training set of the four minority classes of 'automobile', 'bird', 'cat' and 'horse', to generate fake data for the purpose of balancing each class to a total of 5,000 instances. The models of the generator and discriminator of WGAN-GP for 32x32x3 images of CIFAR-10 are described in Figures 4.8(a) and 4.8(b).



```
[<tf.Variable 'gen/dense/kernel:0' shape=(100, 32768) dtype=float32_ref>,
 <tf.Variable 'gen/dense/bias:0' shape=(32768,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization/gamma:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization/beta:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/conv2d_transpose/kernel:0' shape=(5, 5, 128, 128) dtype=float32_ref>,
 <tf.Variable 'gen/conv2d_transpose/bias:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization_1/gamma:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization_1/beta:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/conv2d_transpose_1/kernel:0' shape=(5, 5, 128, 128) dtype=float32_ref>,
 <tf.Variable 'gen/conv2d_transpose_1/bias:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization_2/gamma:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization_2/beta:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/conv2d_transpose_2/kernel:0' shape=(5, 5, 128, 128) dtype=float32_ref>,
 <tf.Variable 'gen/conv2d_transpose_2/bias:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization_3/gamma:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization_3/beta:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/conv2d_transpose_3/kernel:0' shape=(5, 5, 128, 128) dtype=float32_ref>,
 <tf.Variable 'gen/conv2d_transpose_3/bias:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization_4/gamma:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization_4/beta:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'gen/conv2d_transpose_4/kernel:0' shape=(5, 5, 3, 128) dtype=float32_ref>,
 <tf.Variable 'gen/conv2d_transpose_4/bias:0' shape=(3,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization_5/gamma:0' shape=(3,) dtype=float32_ref>,
 <tf.Variable 'gen/batch_normalization_5/beta:0' shape=(3,) dtype=float32_ref>,
 <tf.Variable 'gen/dense_1/kernel:0' shape=(3072, 3072) dtype=float32_ref>,
 <tf.Variable 'gen/dense_1/bias:0' shape=(3072,) dtype=float32_ref>]
```

(a) Generator Network



```
[<tf.Variable 'dis/conv2d/kernel:0' shape=(5, 5, 3, 128) dtype=float32_ref>,
 <tf.Variable 'dis/conv2d/bias:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'dis/conv2d_1/kernel:0' shape=(5, 5, 128, 128) dtype=float32_ref>,
 <tf.Variable 'dis/conv2d_1/bias:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'dis/conv2d_2/kernel:0' shape=(5, 5, 128, 128) dtype=float32_ref>,
 <tf.Variable 'dis/conv2d_2/bias:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'dis/conv2d_3/kernel:0' shape=(5, 5, 128, 128) dtype=float32_ref>,
 <tf.Variable 'dis/conv2d_3/bias:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'dis/batch_normalization/gamma:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'dis/batch_normalization/beta:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'dis/dense/kernel:0' shape=(32768, 1) dtype=float32_ref>,
 <tf.Variable 'dis/dense/bias:0' shape=(1,) dtype=float32_ref>]
```

(b) Discriminator Network

Figure 4.8: WGAN-GP Model Description for CIFAR-10 Dataset

Since CIFAR-10 contains more complex images than FMNIST, the generator would typically need to be trained for a lot more epochs to be able to learn the features effectively and start generating realistic looking images as outputs. The generated images sampled at arbitrary epochs of WGAN-GP training phase for the class 'horse' can be observed in Figure 4.9.

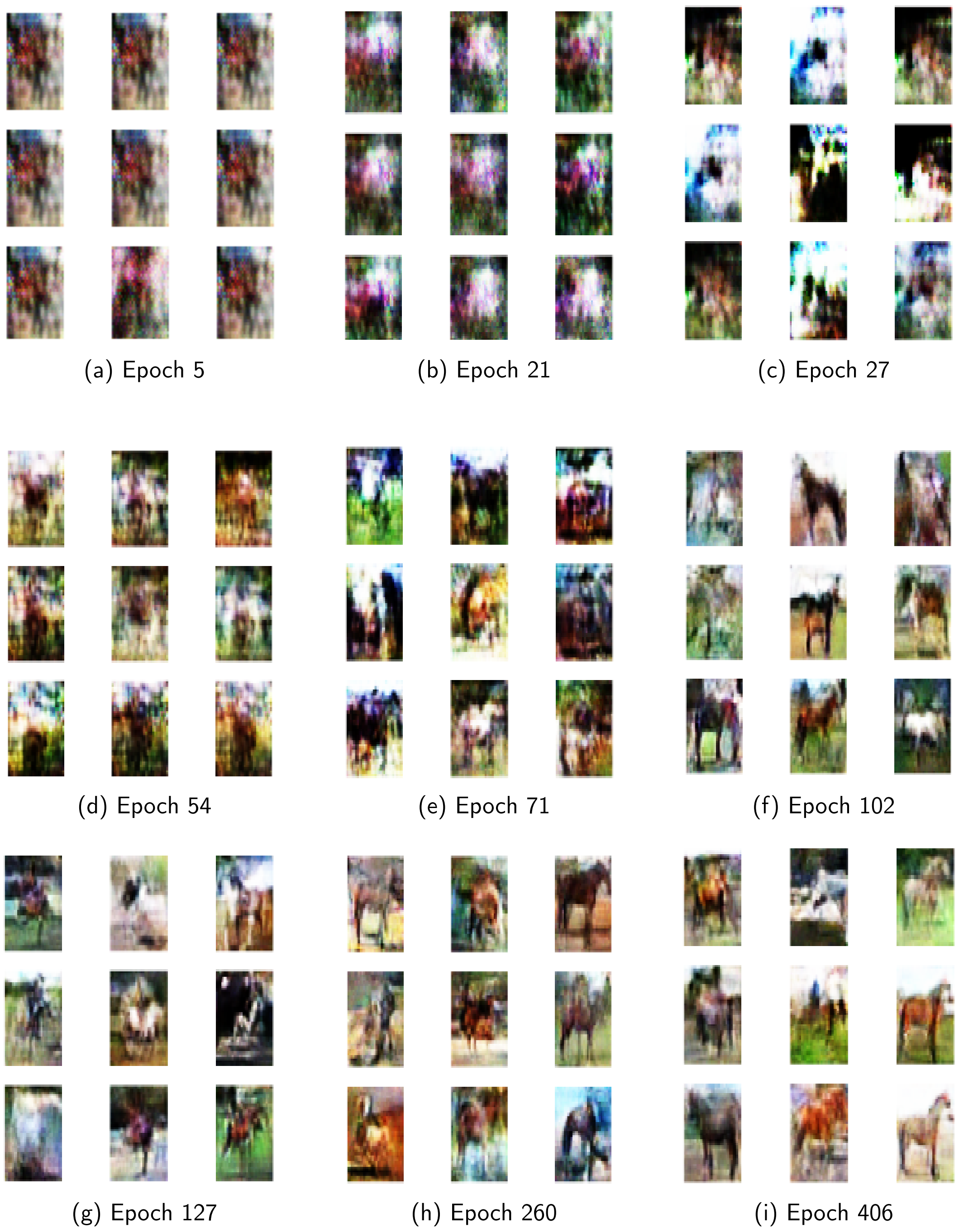


Figure 4.9: Data Generated for the Minority class 'Horse' in the Imbalanced CIFAR-10 using WGAN-GP

Starting from training epoch 406, the generator starts generating realistic-looking image outputs for the minority class 'horse'.

4.4 Classification Architecture

The classification model is implemented in Keras. Based on the design choices described in Chapter 3, the hybrid CNN-SVM architecture is constructed for the two datasets separately, since they have different image dimensions and hence would have different layer dimensions as well.

The final layer of the trained CNN model is replaced with SVM as the prediction layer. The CNN is then trained via SVM through cross-validation.

As highlighted in Chapter 3, SVM has a time complexity of *$O(n^2)$* or higher, making it computationally complex and expensive. Therefore, in our implementation, we have randomly sampled 1000 instances from each class (in both datasets) in case of balanced datasets, and proportionately reduced the number of samples of imbalanced classes in the scenario of imbalanced datasets. The details of the number of instances sampled per class are specified in the following subsections.

Apart from the three datasets mentioned in Figure 3.8, the classifier is also trained on the original, balanced datasets of FMNIST and CIFAR-10.

4.4.1 CNN Model Description for FMNIST

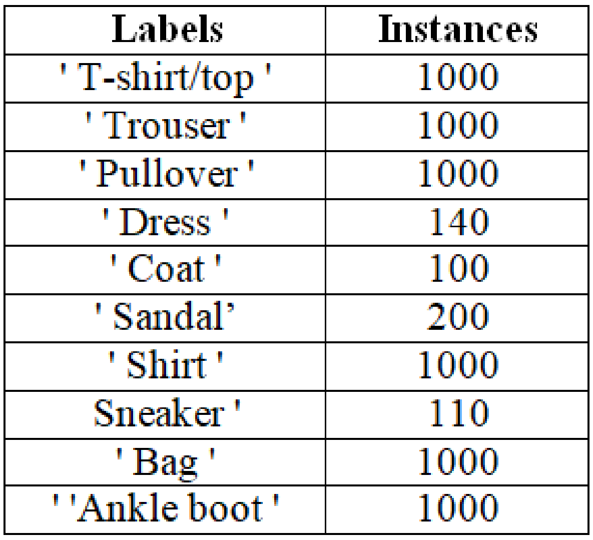
The CNN model for FMNIST dataset, which takes images of dimensions (28 x 28 x 1) as input, can be observed below:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_15 (Conv2D)           (None, 28, 28, 64)        640
_________________________________________________________________
activation_21 (Activation)   (None, 28, 28, 64)        0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 28, 28, 64)        36928
_________________________________________________________________
activation_22 (Activation)   (None, 28, 28, 64)        0
_________________________________________________________________
max_pooling2d_7(MaxPooling2) (None, 14, 14, 64)        0
_________________________________________________________________
dropout_11 (Dropout)         (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 14, 14, 128)       73856
```

```
_________________________________________________________________
activation_23 (Activation)   (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 14, 14, 128)       147584
_________________________________________________________________
activation_24 (Activation)   (None, 14, 14, 128)       0
_________________________________________________________________
max_pooling2d_8(MaxPooling2) (None, 7, 7, 128)         0
_________________________________________________________________
dropout_12 (Dropout)         (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 7, 7, 256)         295168
_________________________________________________________________
activation_25 (Activation)   (None, 7, 7, 256)         0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 7, 7, 256)         590080
_________________________________________________________________
activation_26 (Activation)   (None, 7, 7, 256)         0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 7, 7, 256)         590080
_________________________________________________________________
activation_27 (Activation)   (None, 7, 7, 256)         0
_________________________________________________________________
max_pooling2d_9(MaxPooling2) (None, 3, 3, 256)         0
_________________________________________________________________
dropout_13 (Dropout)         (None, 3, 3, 256)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_7 (Dense)              (None, 1024)              2360320
_________________________________________________________________
activation_28 (Activation)   (None, 1024)              0
_________________________________________________________________
dropout_14 (Dropout)         (None, 1024)              0
_________________________________________________________________
dense_8 (Dense)              (None, 1024)              1049600
_________________________________________________________________
activation_29 (Activation)   (None, 1024)              0
_________________________________________________________________
```

```
dropout_15 (Dropout)         (None, 1024)            0
_________________________________________________________________
dense_9 (Dense)              (None, 10)              10250
_________________________________________________________________
activation_30 (Activation)   (None, 10)              0
=================================================================
```

The distribution for sampling instances from the imbalanced training dataset for SVM can be observed in Table 4.3. The test set also consists of a total of 6500 instances (reduced from 10,000) to match the training set size, however, in that case, the number of instances in each class is kept equal to reflect balanced classes.



Labels	Instances
' T-shirt/top '	1000
' Trouser '	1000
' Pullover '	1000
' Dress '	140
' Coat '	100
' Sandal'	200
' Shirt '	1000
Sneaker '	110
' Bag '	1000
' 'Ankle boot '	1000

Table 4.3: Data Distribution of Training Data Sampled from Imbalanced FMNIST for SVM

4.4.2 CNN Model Description for CIFAR-10

The CIFAR-10 dataset contains images of dimensions (32 x 32 x 3), and thus requires different dimensions of layers than FMNIST. The CNN model structure for CIFAR-10 classification is given below:

```
Layer (type)                 Output Shape            Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 64)      1792
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 64)      0
_________________________________________________________________
```

```
conv2d_2 (Conv2D)            (None, 32, 32, 64)        36928
_________________________________________________________________
activation_2 (Activation)    (None, 32, 32, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
activation_3 (Activation)    (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 128)       147584
_________________________________________________________________
activation_4 (Activation)    (None, 16, 16, 128)       0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 256)         295168
_________________________________________________________________
activation_5 (Activation)    (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 256)         590080
_________________________________________________________________
activation_6 (Activation)    (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 8, 8, 256)         590080
_________________________________________________________________
activation_7 (Activation)    (None, 8, 8, 256)         0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 256)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              4195328
```

```
_________________________________________________________________
activation_8 (Activation)    (None, 1024)              0
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              1049600
_________________________________________________________________
activation_9 (Activation)    (None, 1024)              0
_________________________________________________________________
dropout_5 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                10250
_________________________________________________________________
activation_10 (Activation)   (None, 10)                0
=================================================================
```

The distribution for sampling instances from the imbalanced training dataset for SVM can be observed in Table 4.4. As in FMNIST, the test set size is also reduced to 6500, with each class having equal representation to reflect class balance.

Labels	**Instances**
'airplane'	1000
'automobile'	500
'bird'	300
'cat'	200
'deer'	1000
'dog'	1000
'frog'	1000
'horse'	100
'ship'	1000
'truck'	1000

Table 4.4: Data Distribution of Training Data Sampled from Imbalanced CIFAR-10 for SVM

5 Results and Discussion

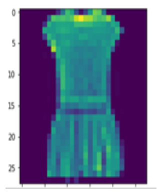
This chapter presents the results obtained in the intermediate steps of the study as well as the final values of the classification metrics obtained for each of the datasets. These results are compared and contrasted and critically analysed to derive insights regarding the research question being addressed.

5.1 Data Generation using Image Transforms

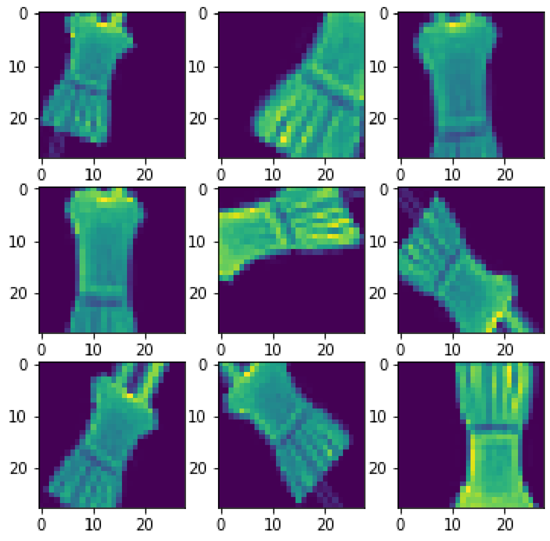
An example of images generated by applying geometric and photo-metric transforms on a sample of minority class 'Dress' of FMNIST dataset can be observed in Figure 5.1. The same for a sample of class 'cat' from CIFAR-10 can be observed in Figure 5.3.

This method of image data augmentation is simple and intuitive, and results in fairly representative label-preserving samples. However, note that this method has certain limitations, which can be explained using Figure 5.1:

- Notice that the bottom right image as well as the second image in the first row in Figure 5.1(b) are not representative of a dress anymore. We can observe loss of context in this case
- The first image in second row of 5.1(b) looks quite similar to some data samples of 'T-Shirt' class (Refer to Figure 5.2), and if too many such instances are generated, it can cause wrong features to be learned
- Since the basic image remains the same, this method fails to introduce a good variety in the dataset, hence restricting the amount of features that can be learned, or possibly causing overfitting in extreme cases



(a) Original Image



(b) New images generated using Image Transformations

Figure 5.1: An example of Generating New Data using Image Transforms on 'Dress' sample of FMNIST dataset

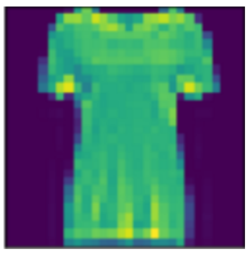
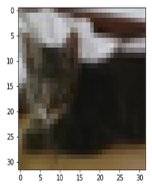
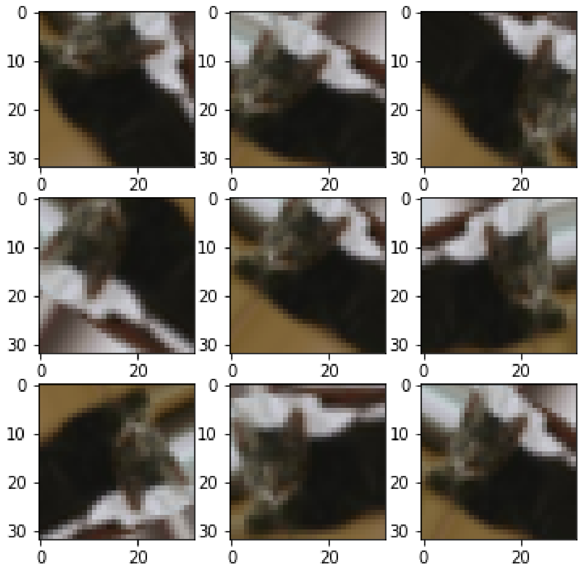


Figure 5.2: A sample of 'T-Shirt' class from FMNIST, which looks similar to some of the images in 5.1(b)



(a) Original Image



(b) New images generated using Image Transformations

Figure 5.3: An example of Generating New Data using Image Transforms on 'cat' sample of CIFAR-10

Note that the images generated for 'cat' data sample from CIFAR-10 (Figure 5.3) do not exhibit such issues to the same extent (apart from the lack of variability). This is because the images in CIFAR-10 are semantically more complex, and hence easier to tell apart from one another. However, it is still possible to generate data that is not truly representative of the given class, such as an instance when the image of a 'cat' looks similar to that of a 'dog' (Refer to data samples of 'dog' class of CIFAR-10 in Figure 5.4).

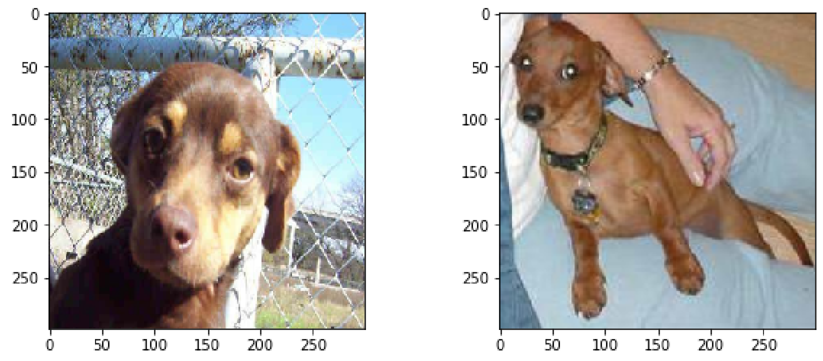
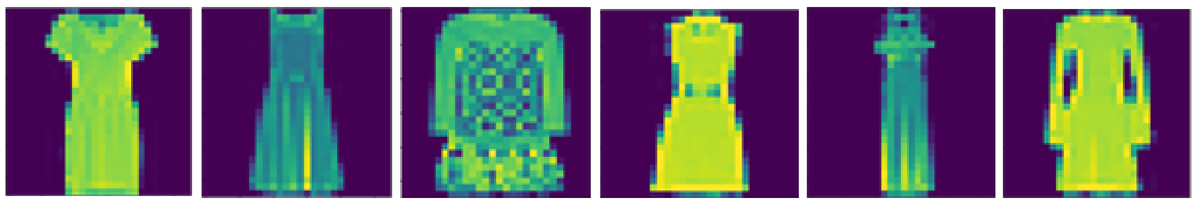


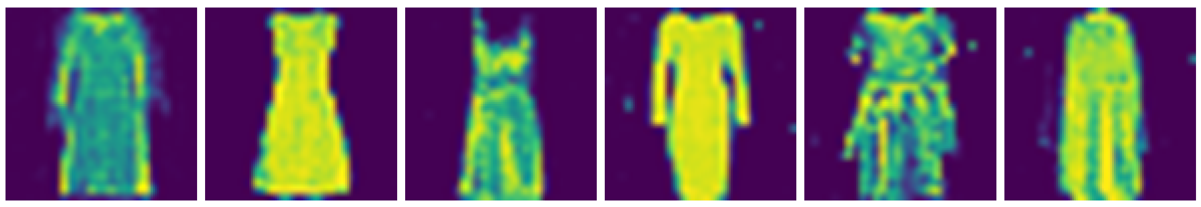
Figure 5.4: Images from the class 'dog' of CIFAR-10 which may look similar to some 'cat' images generated

5.2 Data Generation using WGAN-GP

Some of the data samples generated for the 'Dress' minority class of imbalanced FMNIST dataset are shown in Figure 5.5 (b), and can be contrasted with Figure 5.5 (a), which displays randomly sampled data points from the 'Dress' class of original FMNIST dataset. The same can be observed for the 'cat' class of imbalanced CIFAR-10 dataset, where the real and fake images can be observed in Figures 5.6(a) and (b).

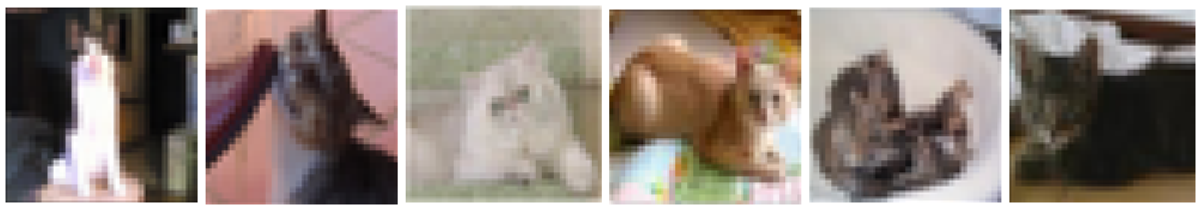


(a) Original Images

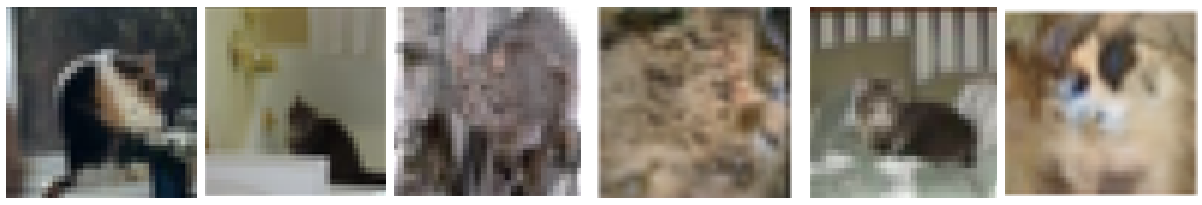


(b) Fake Images Generated using WGAN-GP

Figure 5.5: Comparison of Real Images and Fake Images generated using WGAN-GP for 'Dress' class of FMNIST



(a) Original Images



(b) Fake Images Generated using WGAN-GP

Figure 5.6: Comparison of Real Images and Fake Images generated using WGAN-GP for 'cat' class of CIFAR-10

The Figures 5.5 and 5.6 is a good example of how the 'real' and 'fake' images are mixed for the 'Real or Fake?' test for determining how realistic the generated images are to a human subject. I conducted this experiment on myself, and labeled all of the generated images for FMNIST as 'real'.

Since CIFAR-10 consists of images of real objects, some of which are close in appearance (such as 'cat' and 'dog'), the lower resolution of the images makes the identification process a bit difficult. As an example, I correctly labeled one 'fake' sample as 'fake' from the 'cat' class. However, the reason for making that decision was that the image was too zoomed in and blurred, making it look more like noise. Apart from that, I correctly labeled 2 instances from the 'bird' class as fake, as they lacked a body structure. For the other two classes ('automobile' and 'horse'), all samples were labeled as 'real'.

Note that the generated images demonstrate the following qualities:

- Realistic looking, and impossible to tell apart from the original dataset images of the same class by a human observer
- Preservation of context or semantic information of the class in question, meaning that the features of the cat class from the real images have been efficiently learned through this architecture
- Samples are not repititive, signifying that there is no overfitting

- Samples generated are variable or diverse in nature, therefore resulting in an efficiently augmented dataset which contains samples representing many possibilities

5.3 Classification Results Obtained

This section summarizes the results obtained for Validation[1] Accuracy, Precision, Recall and F1-Score of the classifier used for the variations of the two datasets considered. As mentioned in Chapter 3 (Figure 3.8), the four variations of each dataset considered are: 'Original Dataset (Balanced)', 'Imbalanced' (after removing instances of some classes), 'Balanced using Image Transforms' (resulting from data augmentation for minority classes using images generted by geometric and photo-metric transforms) and 'Balanced using WGAN-GP' (resulting from data augmentation for minority classes using images generated by WGAN-GP). The classifier is separately trained on each of these datasets and these trained models are used for evaluation. However, as mentioned in Chapter 4, no change is made to the test set of these datasets.

The results for FMNIST and CIFAR-10 will be documented in different sub-sections, and the overall observations will be analysed and discussed in the subsequent section of this chapter.

Classification Results for FMNIST

The performance of the classifier on FMNIST is evaluated using the testing metrics recorded in Table 5.1.

It becomes obvious that imbalance causes the classifier's performance to go down, with the Accuracy being reduced by 5.25% , and the F1-Score going down by the same amount as well.

The effect of this imbalance is reduced in the datasets balanced using data augmentation techniques. While using traditional image transforms results in an improvement of approximately 4% in testing Accuracy, and of around 4.3% in the F1-Score, it can be observed that WGAN-GP does a much better job and increases the Accuracy and F1-Score by 5% and 4.5% in comparison to the imbalanced dataset.

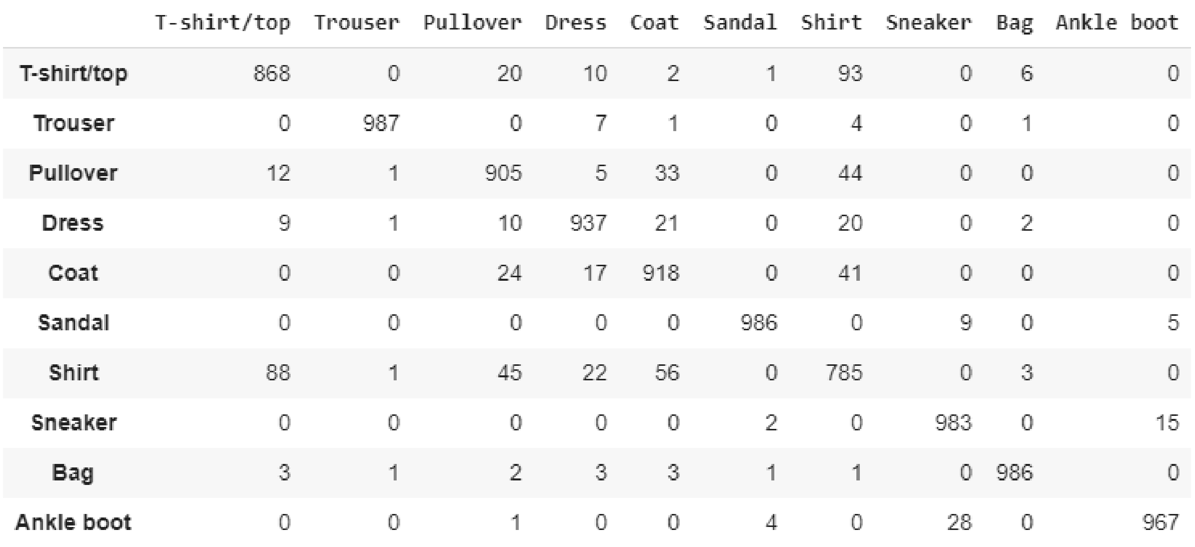
[1]Note that "Validation" and "Testing" are used interchangeably in this study.

Dataset	Accuracy	Precision	Recall	F1-Score
Original (Balanced)	0.932	0.931	0.932	0.932
Imbalanced	0.883	0.897	0.884	0.883
Balanced using Image Transforms	0.919	0.922	0.92	0.921
Balanced using WGAN-GP	0.928	0.925	0.921	0.923

Table 5.1: Testing Metrics for Classification of Variations of FMNIST

Although these metrics represent the overall classifier performance, a much more intuitive and informative way to study the effects would be to observe the confusion matrices for each case. The relevant classes to study for us would be the ones which we create an imbalance in, that is, 'Dress', 'Coat', 'Sandal' and 'Sneaker'.

For the original dataset (Figure 5.7), we can observe that the majority of the test samples are correctly classified.

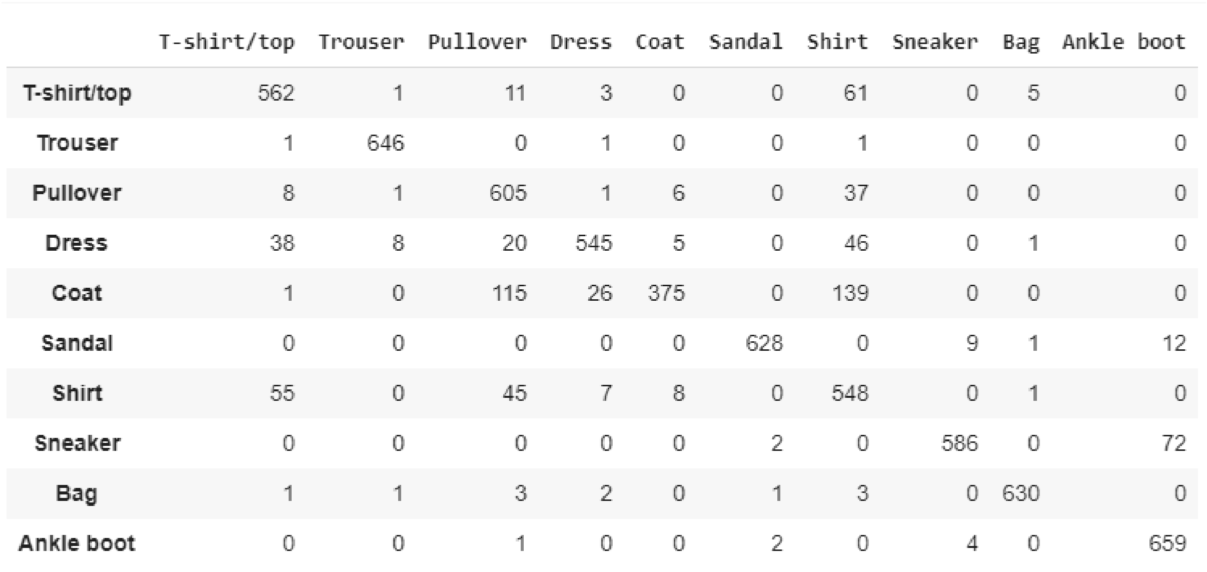


	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	868	0	20	10	2	1	93	0	6	0
Trouser	0	987	0	7	1	0	4	0	1	0
Pullover	12	1	905	5	33	0	44	0	0	0
Dress	9	1	10	937	21	0	20	0	2	0
Coat	0	0	24	17	918	0	41	0	0	0
Sandal	0	0	0	0	0	986	0	9	0	5
Shirt	88	1	45	22	56	0	785	0	3	0
Sneaker	0	0	0	0	0	2	0	983	0	15
Bag	3	1	2	3	3	1	1	0	986	0
Ankle boot	0	0	1	0	0	4	0	28	0	967

Figure 5.7: Confusion Matrix for Original FMNIST Dataset Classification

From the confusion matrix of the imbalanced dataset[2] (Figure 5.8), it is interesting to note that despite the imbalance, the accuracy of classification for 'Sandal' and 'Sneaker' is still very high (95% and 88.7%).

However, a visible degradation can be observed for the class 'Coat', for which the least number of training samples were present. Data belonging to 'Coat' class gets misclassified 43% of the times, and it can be seen that most of them are misclassified as 'Pullover' or 'Shirt', which are majority classes, and may have certain similar characteristics to 'Coat'. Thereofore, a bias towards majority classes is prominent in this case.

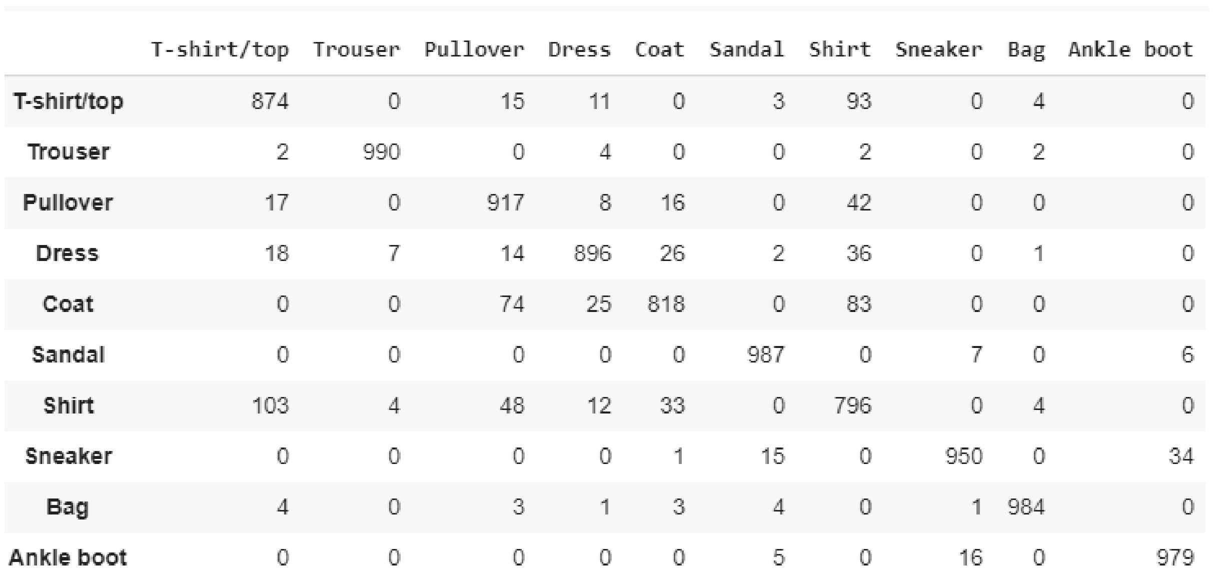


	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	562	1	11	3	0	0	61	0	5	0
Trouser	1	646	0	1	0	0	1	0	0	0
Pullover	8	1	605	1	6	0	37	0	0	0
Dress	38	8	20	545	5	0	46	0	1	0
Coat	1	0	115	26	375	0	139	0	0	0
Sandal	0	0	0	0	0	628	0	9	1	12
Shirt	55	0	45	7	8	0	548	0	1	0
Sneaker	0	0	0	0	0	2	0	586	0	72
Bag	1	1	3	2	0	1	3	0	630	0
Ankle boot	0	0	1	0	0	2	0	4	0	659

Figure 5.8: Confusion Matrix for Imbalanced FMNIST Dataset Classification

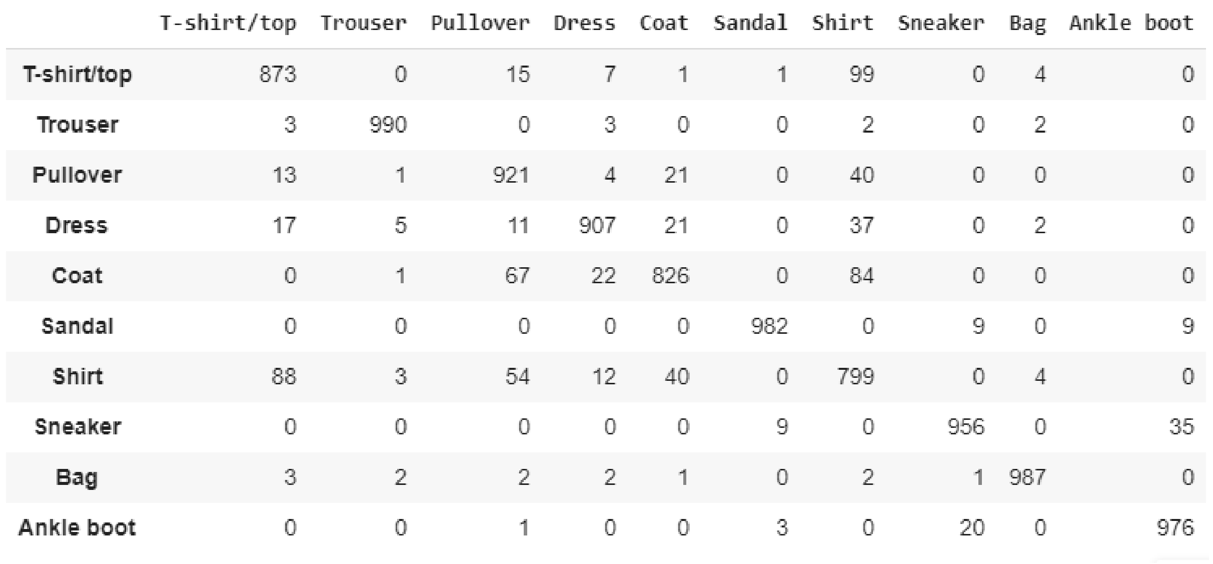
On being balanced using both techniques (Figure 5.9 and Figure 5.10), we can see that the number of correct classifications for each of the imbalanced classes becomes almost the same as it was in the original dataset. However, on an average, the numbers are higher for the dataset augmented using WGAN-GP.

[2]The total number of samples are less per class as the test dataset was reduced to 6400 with nearly equal instances of each class for fitting the SVM better, as mentioned in Chapter 4.



	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	874	0	15	11	0	3	93	0	4	0
Trouser	2	990	0	4	0	0	2	0	2	0
Pullover	17	0	917	8	16	0	42	0	0	0
Dress	18	7	14	896	26	2	36	0	1	0
Coat	0	0	74	25	818	0	83	0	0	0
Sandal	0	0	0	0	0	987	0	7	0	6
Shirt	103	4	48	12	33	0	796	0	4	0
Sneaker	0	0	0	0	1	15	0	950	0	34
Bag	4	0	3	1	3	4	0	1	984	0
Ankle boot	0	0	0	0	0	5	0	16	0	979

Figure 5.9: Confusion Matrix for Classification of FMNIST Dataset Balanced using Image Transforms

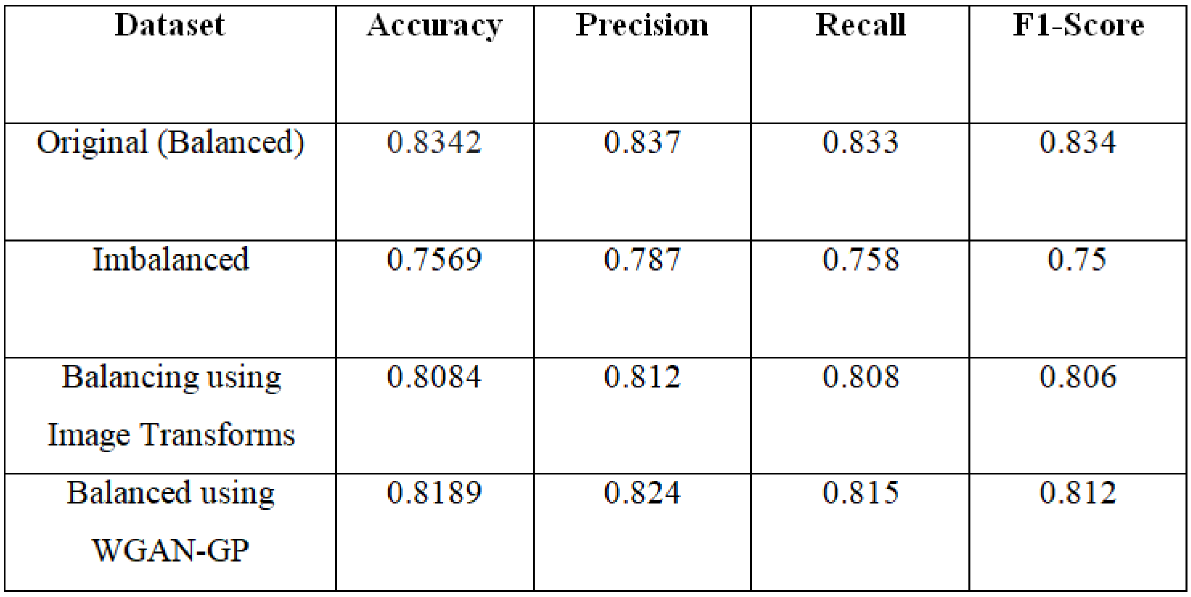


	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	873	0	15	7	1	1	99	0	4	0
Trouser	3	990	0	3	0	0	2	0	2	0
Pullover	13	1	921	4	21	0	40	0	0	0
Dress	17	5	11	907	21	0	37	0	2	0
Coat	0	1	67	22	826	0	84	0	0	0
Sandal	0	0	0	0	0	982	0	9	0	9
Shirt	88	3	54	12	40	0	799	0	4	0
Sneaker	0	0	0	0	0	9	0	956	0	35
Bag	3	2	2	2	1	0	2	1	987	0
Ankle boot	0	0	1	0	0	3	0	20	0	976

Figure 5.10: Confusion Matrix for Classification of FMNIST Dataset Balanced using WGAN-GP

Classification Results for CIFAR-10

The performance of the classifier on FMNIST is evaluated using the testing metrics recorded in Table 5.1. It can be observed that the negative impact of class imbalance in this case is higher than observed in FMNIST, with a drop of 9.2% in Testing Accuracy, and 10.07% in F1-Score. The improvement observed as a result of data augmentation is also higher as compared to FMNIST. Using geometric and photo-metric transforms, an increase of 6.8% in Accuracy and of 7.5% in F1-Score is observed. This is much higher when the dataset is balanced using images generated by WGAN-GP, with an increase of 8.2% and 8.26%, in comparison to the imbalanced dataset.

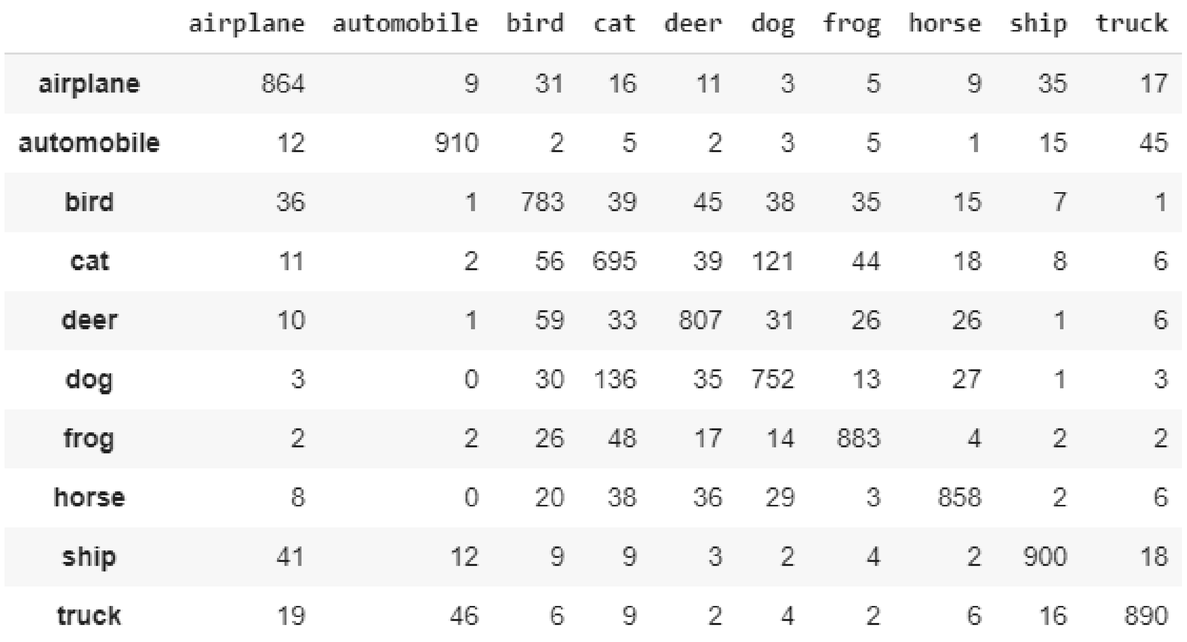


Dataset	Accuracy	Precision	Recall	F1-Score
Original (Balanced)	0.8342	0.837	0.833	0.834
Imbalanced	0.7569	0.787	0.758	0.75
Balancing using Image Transforms	0.8084	0.812	0.808	0.806
Balanced using WGAN-GP	0.8189	0.824	0.815	0.812

Table 5.2: Testing Metrics for Classification of Variations of CIFAR-10

While studying the confusion matrices obtained for the different classification tasks performed, we will focus on the four imbalanced classes in particular, which are 'cat', 'automobile', 'horse' and 'bird'.

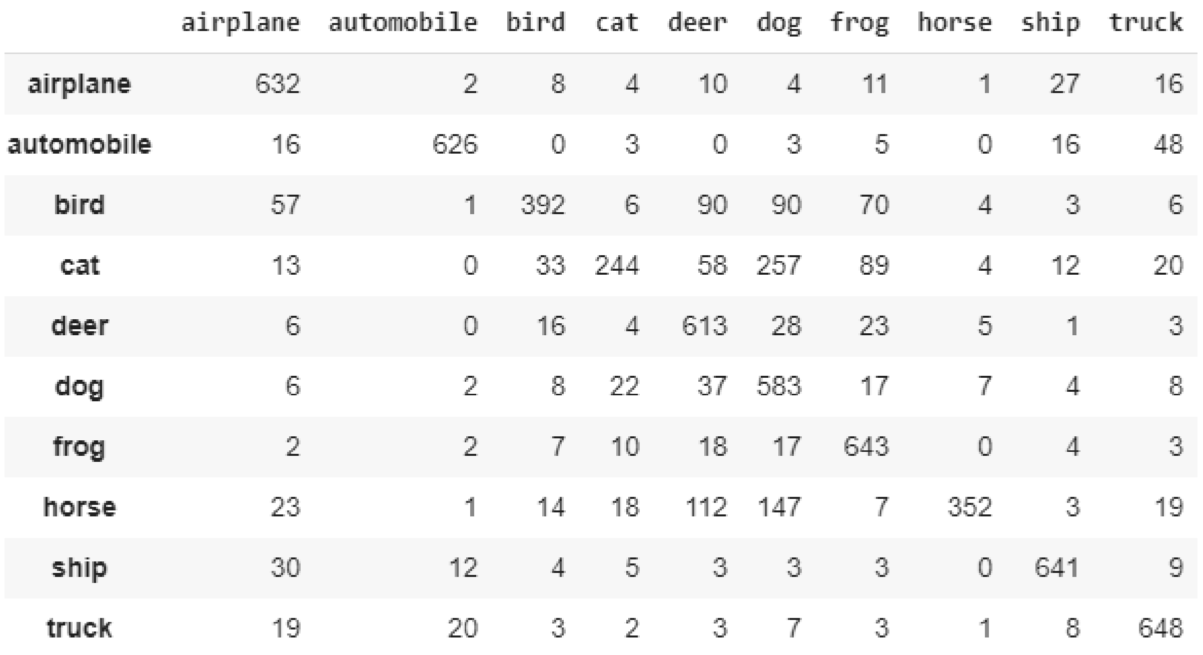
From Figure 5.11, we can observe that the instances of 'cat' and 'bird' classes are only correctly classified 69.5% and 78.3% of the times. Therefore, to observe the impact of imbalance, this needs to be kept in mind.



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	864	9	31	16	11	3	5	9	35	17
automobile	12	910	2	5	2	3	5	1	15	45
bird	36	1	783	39	45	38	35	15	7	1
cat	11	2	56	695	39	121	44	18	8	6
deer	10	1	59	33	807	31	26	26	1	6
dog	3	0	30	136	35	752	13	27	1	3
frog	2	2	26	48	17	14	883	4	2	2
horse	8	0	20	38	36	29	3	858	2	6
ship	41	12	9	9	3	2	4	2	900	18
truck	19	46	6	9	2	4	2	6	16	890

Figure 5.11: Confusion Matrix for the Classification of Original CIFAR-10

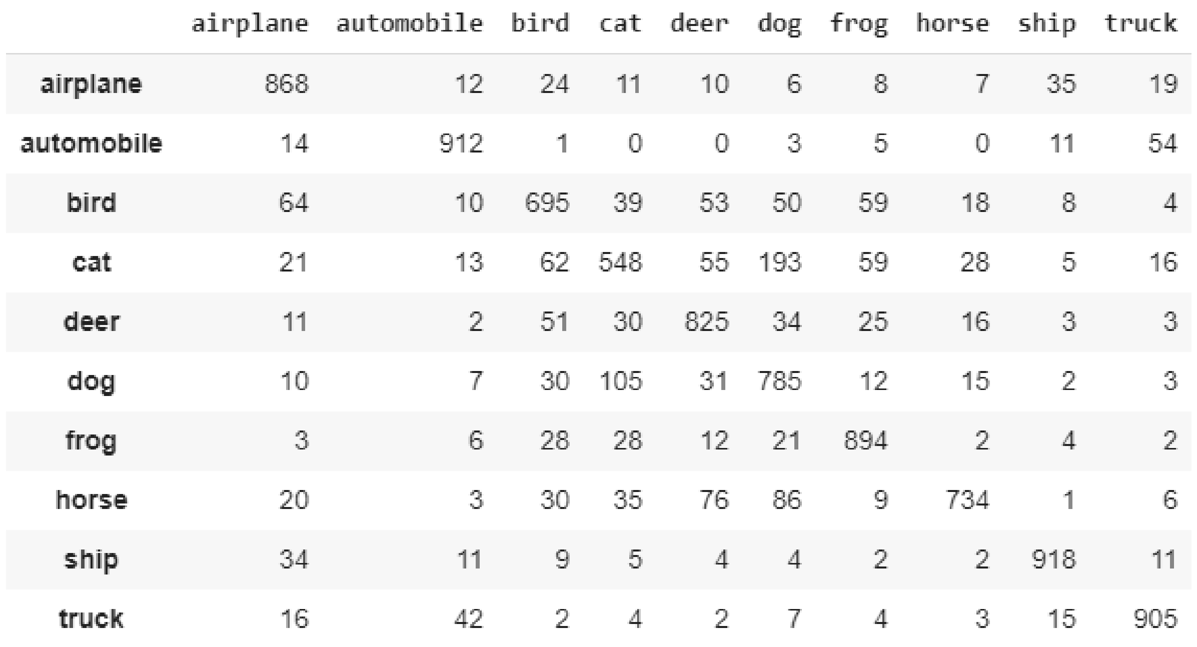
The impact of imbalance can be observed in Figure 5.12. The most visible difference can be observed for classes 'bird', 'cat' and 'horse', where only 48.45%, 33.42% and 50.5% of the data samples are correctly classified. Further, it can be observed for class 'cat' that the majority of misclassified samples are classified as 'dog', which indicates a bias towards that majority class. It is also interesting to note that for the original dataset, 10.5% of the samples for 'dog' class were classified as 'cat', however, in the imbalanced dataset, only 3% of 'dog' data samples are wrongly labeled as 'cat'. This is also a reflection of how minority classes have little contribution to the overall features learned.



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	632	2	8	4	10	4	11	1	27	16
automobile	16	626	0	3	0	3	5	0	16	48
bird	57	1	392	6	90	90	70	4	3	6
cat	13	0	33	244	58	257	89	4	12	20
deer	6	0	16	4	613	28	23	5	1	3
dog	6	2	8	22	37	583	17	7	4	8
frog	2	2	7	10	18	17	643	0	4	3
horse	23	1	14	18	112	147	7	352	3	19
ship	30	12	4	5	3	3	3	0	641	9
truck	19	20	3	2	3	7	3	1	8	648

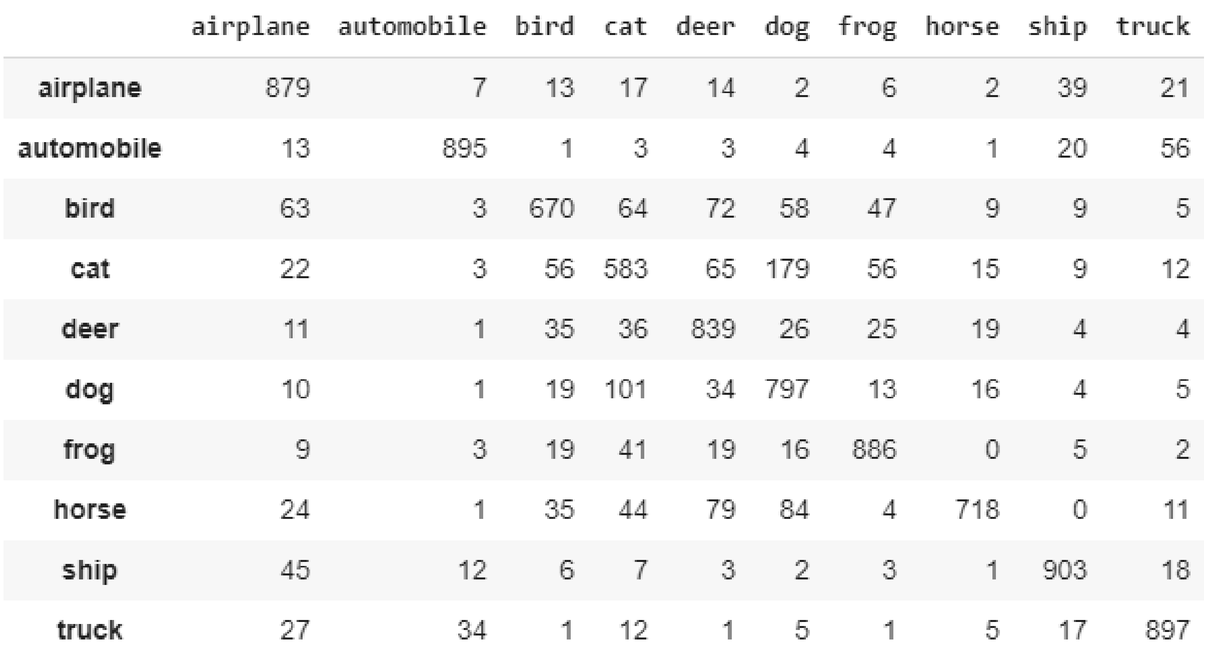
Figure 5.12: Confusion Matrix for Imbalanced CIFAR-10 Dataset Classification

After balancing the dataset using data augmentation, from both Figures 5.13 and 5.14, it can be observed that the results are similar to Figure 5.11 for the original dataset. As in the case of FMNIST, on an average, the number of correct predictions for each of the imbalanced classes is more when data augmentation is done using WGAN-GP.



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	868	12	24	11	10	6	8	7	35	19
automobile	14	912	1	0	0	3	5	0	11	54
bird	64	10	695	39	53	50	59	18	8	4
cat	21	13	62	548	55	193	59	28	5	16
deer	11	2	51	30	825	34	25	16	3	3
dog	10	7	30	105	31	785	12	15	2	3
frog	3	6	28	28	12	21	894	2	4	2
horse	20	3	30	35	76	86	9	734	1	6
ship	34	11	9	5	4	4	2	2	918	11
truck	16	42	2	4	2	7	4	3	15	905

Figure 5.13: Confusion Matrix for Classification of CIFAR-10 Dataset Balanced using Image Transforms



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	879	7	13	17	14	2	6	2	39	21
automobile	13	895	1	3	3	4	4	1	20	56
bird	63	3	670	64	72	58	47	9	9	5
cat	22	3	56	583	65	179	56	15	9	12
deer	11	1	35	36	839	26	25	19	4	4
dog	10	1	19	101	34	797	13	16	4	5
frog	9	3	19	41	19	16	886	0	5	2
horse	24	1	35	44	79	84	4	718	0	11
ship	45	12	6	7	3	2	3	1	903	18
truck	27	34	1	12	1	5	1	5	17	897

Figure 5.14: Confusion Matrix for Classification of CIFAR-10 Dataset Balanced using Images Generated by WGAN-GP

Changes related to precision, recall and F1-Score can also be observed in this class-by-class manner using the confusion matrices.

5.3.1 Analysis and Discussion

From the results obtained in this study, it is observed that class imbalance has a detrimental impact on the performance of a classifier.The data belonging to minority classes get mislabeled as instances of the classes which are in majority.

However, it was observed in some minority classes, that despite the imbalance, most of their samples were being classified correctly. This was more eminent in FMNIST than CIFAR-10, and can be attributed to the fact that all the classes are non-overlapping and perfectly separable[3]. The classifier used in this study was strong enough to learn the features from less data as well, as the images are relatively simple. In such cases, it might be possible to maintain the classifier's performance without performing any data augmentation at all.

It is fair to say that as dataset complexity increases, the impact of class imbalance also increases.

[3]Although the CIFAR-10 dataset is also composed of non-overlapping classes, there are some similarities between certain classes such as 'automobile' and 'truck', which can often be a cause of confusion.

Another noticeable fact is that WGAN-GP generates images that are much more realistic, relevant and diverse, as compared to the training set obtained when additional images are generated by applying image transformations. Therefore, WGAN-GP results in superior quality training dataset, which ultimately results in better classification performance.

6 Conclusion

The task of image classification is a vital component of many day to day applications. The occurrence of imbalance in datasets collected from real-life domains is unavoidable due to numerous factors. It has been established that this imbalance has a detrimental effect on the performance of classifiers. In this research we explored the traditional methods of data augmentation to create balance in imbalanced datasets, by using geometric and photo-metric transformations, which led to an observable improvement in the performance of the Hybrid CNN-SVM Classifier. Using this as a baseline, we then generated new samples for the minority classes of the imbalanced datasets using an improved variation of the Generative Adversarial Network, called Wasserstein Generative Adversarial Network using Gradient Penalty. This method of dataset augmentation outperforms the traditional method in terms of the quality and variability of the samples generated, which eventually leads to a significant improvement in the classifier performance. It is also important to note that the performance improvement becomes more noticeable as the complexity of the dataset increases. Thus, it can be concluded that Generative Adversarial Networks provide a more advanced and promising data augmentation solution for the class imbalance problem in datasets and can increase the efficiency of classification models in such scenarios.

However,it is important to understand if the effects observed are specifically due to imbalance in dataset, or are simply a result of the overall reduction in quantity of data. This is a question that can be further explored. Certain other limitations of this study, and the methods proposed to overcome them, have been outlined in the section below.

6.1 Limitations and Future Work

In the course of implementation and evaluation of this research, certain limitations were noticed. This section describes them, with suggestions for extending this research and overcoming them.

In this research, we have identified the imbalanced classes in a dataset by manual inspection and trained the WGAN-GP architecture class-by-class, to generate more instances of each specific class in question. This approach is not sufficient for practical applications due to the need for human involvement, and also because usually, minority classes have scarce data samples, which makes it challenging to train a GAN for the generation of new images.

To overcome this issue, an autoencoder-based initialization strategy can be adopted, as seen in [67], in which the adversarial training is done on the entire dataset at once, i.e., both minority and majority classes. This enables the GAN architecture to extract general features for the specific dataset, and then apply class conditioning on the latent space [87, 88], which helps the autoencoder learn what the generative model inputs should be like for different classes, and help drive the image generation process towards the desired classes.

Another reason why the WGAN-GP training process adopted in this research requires human intervention is because the images being generated during each epoch are deemed as "realistic" or "not realistic" based on the perception of the researcher. Therefore, the number of training epochs cannot be pre-determined and the outputs should be monitored and visually inspected. However, a number of GAN evaluation metrics [89] can be leveraged to automate this process and let the GAN determine the quality of the output mathematically. Some such metrics are inception score [90], GAN Quality Index [91], etc. The SSIM metric introduced in Chapter 3 can also be implemented to study the diversity and variability of the metrics generated.

It would also be interesting to experiment with different GAN architectures such as Auxiliary Classifier GAN [92], Conditional GAN [55], Balancing GAN [67], etc. to understand if that has an impact on the results.

Another important thing to note is that this research only considers imbalance in terms of skewed class distribution, which is given by the imbalance ratio in Equation (1). However, it would also be important to test our hypotheses on a range of values of imbalance ratio, as the results can be poor for both high and low values of imbalance ratio [93] and it would help us to understand the pattern of behaviour for the same.

In this research we have only experimented with one classifier architecture. To extend this research, it would be important to experiment with more complex deep neural networks such as ResNet [94] which have been proven as more efficient in image classification of balanced datasets. Additional classification metrics such as Balanced Accuracy [95], G-Mean [96] and most importantly, Area under the ROC (Receiving Operators Characteristics) Curve [97] which plots the true positive rate over false positive rates, providing a visualization to depict the trade-off between correctly classified positive samples and incorrectly classified negative samples.

Further, it has been identified that the skewed distribution by itself may not be the only factor hindering the learning process [34, 98]. Therefore, it would also be important to consider the additional factors that may have a detrimental effect on the classification results on an imbalanced dataset [29]. Some of the characteristics that can be taken into account for a more comprehensive study are:

- The problem of class overlapping or class separability of a dataset [99, 100]
- Impact of presence of noisy data in imbalanced datasets [101, 102], which has been shown to have a greater impact on minority classes than usual cases [103]
- Dataset shift problem [104, 105], which is the case where training data and test data follow different distributions, and can often occur due to bias in sample selection
- Impact of borderline sample points which are located in the areas surrounding class boundaries, where minority and majority classes overlap [41, 106]
- Effect of lack of information or density in training data [107], where the small dataset size is not sufficient to learn a generalized distribution and leads to model overfitting

It would also be a good idea to conduct the experiments on bigger and more complex datasets such as Imagenet [108] to validate how increase in dataset size and complexity affects the classification results. This would also give us an opportunity to use techniques such as Data Undersampling and draw a comparison with the methods considered by us.

Last but not the least, it may be interesting to experiment on datasets which are naturally imbalanced, such as GTSRB [109], which is a traffic sign recognition dataset, and may be beneficial in smart city applications such as self-driving cars. It may also be beneficial to extend the research beyond image datasets to numerical and textual data. The KEEL dataset repository [110] also provides a number of imbalanced datasets with varying imbalance ratios and have pre-processed them using a variety of proven techniques such as Synthetic Minority Over-Sampling Technique (SMOTE) [111], Borderline [112], Safe Levels [113], etc., which would help in comparing and evaluating the effectiveness of GANs as a tool to combat class imbalance.

Bibliography

[1] Michael Zimmerman, IBM THINK Blog. `https://www.ibm.com/blogs/think/2016/05/weathering-hurricane-season-with-cognitive-iot/`.

[2] Pedro Lopez-Garcia, Antonio D. Masegosa, Enrique Onieva, and Eneko Osaba. Ensemble and fuzzy techniques applied to imbalanced traffic congestion datasets: A comparative study. *Lecture Notes in Computer Science*, pages 185–196, 2018. doi: 10.1007/978-3-319-91641-5_16.

[3] Nitesh V. Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Editorial: Special issue on learning from imbalanced data sets. *ACM SIGKDD Explorations Newsletter*, 6(1): 1, 2004. doi: 10.1145/1007730.1007733.

[4] Rehan Akbani, Stephen Kwek, and Nathalie Japkowicz. Applying support vector machines to imbalanced datasets. *Machine Learning: ECML 2004*, . doi: 10.1007/978-3-540-30115-8_7.

[5] Rehan Akbani, Stephen Kwek, and Nathalie Japkowicz. Applying support vector machines to imbalanced datasets. *Machine Learning: ECML 2004*, . doi: 10.1007/978-3-540-30115-8_7.

[6] Scott H. Clearwater and Eric G. Stern. A rule-learning program in high energy physics event classification. *Computer Physics Communications*, 67(2):159–182, 1991. doi: 10.1016/0010-4655(91)90014-c.

[7] Sarah J Graves, Gregory P Asner, Roberta E Martin, Christopher B Anderson, Matthew S Colgan, Leila Kalantari, and Stephanie A Bohlman. Tree species abundance predictions in a tropical agricultural landscape with a supervised classification model and imbalanced data. *Remote Sensing*, 8(2):161, 2016. doi: 10.3390/rs8020161.

[8] Wouter Verbeke, Karel Dejaeger, David Martens, Joon Hur, and Bart Baesens. New insights into churn prediction in the telecommunication sector: A profit driven data mining approach. *European Journal of Operational Research*, 218(1):211–229, 2012. doi: 10.1016/j.ejor.2011.09.031.

[9] Xing-Ming Zhao, Xin Li, Luonan Chen, and Kazuyuki Aihara. Protein classification with imbalanced data. *Proteins: Structure, Function, and Bioinformatics*, 70(4): 1125–1132, 2007. doi: 10.1002/prot.21870.

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

[11] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`.

[13] 2019. URL `https://www.python.org/`.

[14] François Chollet et al. Keras. `https://keras.io`, 2015.

[15] Yoshihiro Shima. Image augmentation for object image classification based on combination of pre-trained CNN and SVM. *Journal of Physics: Conference Series*, 1004:012001, apr 2018. doi: 10.1088/1742-6596/1004/1/012001. URL `https://doi.org/10.1088%2F1742-6596%2F1004%2F1%2F012001`.

[16] Rohith Gandhi. Support vector machine — introduction to machine learning algorithms, 2019. URL `https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47`.

[17] Daniel J. Mashao. Comparing svm and gmm on parametric feature-sets.

[18] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, Berlin, Heidelberg, 1995. ISBN 0-387-94559-8.

[19] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE transactions on Neural Networks*, 13(2):415–425, 2002.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[21] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.

[22] Jianxiong Xiao, James Hays, Krista A. Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010. doi: 10.1109/cvpr.2010.5539970.

[23] B. Mac Namee, P. Cunningham, S. Byrne, and O.I. Corrigan. The problem of bias in training data in regression problems in medical decision support. *Artificial Intelligence in Medicine*, 24(1):51–70, 2002. doi: 10.1016/s0933-3657(01)00092-6.

[24] Wei Wei, Jinjiu Li, Longbing Cao, Yuming Ou, and Jiahang Chen. Effective detection of sophisticated online banking fraud on extremely imbalanced data. *World Wide Web*, 16(4):449–475, July 2013. ISSN 1386-145X. doi: 10.1007/s11280-012-0178-0. URL `http://dx.doi.org/10.1007/s11280-012-0178-0`.

[25] D.A. Cieslak, N.V. Chawla, and A. Striegel. Combating imbalance in network intrusion datasets. *2006 IEEE International Conference on Granular Computing*. doi: 10.1109/grc.2006.1635905.

[26] Dario Martinez. Predicting the improbable, part 1: The imbalanced data problem - datascience.aero, 2017. URL `https://datascience.aero/predicting-improbable-part-1-imbalanced-data-problem/`.

[27] Alberto Fernández, Victoria López, Mikel Galar, María José del Jesus, and Francisco Herrera. Analysing the classification of imbalanced data-sets with multiple classes: Binarization techniques and ad-hoc approaches. *Knowledge-Based Systems*, 42: 97–110, 2013. doi: 10.1016/j.knosys.2013.01.018.

[28] Minlong Lin, Ke Tang, and Xin Yao. Dynamic sampling approach to training neural networks for multiclass imbalance classification. *IEEE Transactions on Neural Networks and Learning Systems*, 24(4):647–660, 2013. doi: 10.1109/tnnls.2012.2228231.

[29] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study1. *Intelligent Data Analysis*, 6(5):429–449, 2002. doi: 10.3233/ida-2002-6504.

[30] Maciej A. Mazurowski, Piotr A. Habas, Jacek M. Zurada, Joseph Y. Lo, Jay A. Baker, and Georgia D. Tourassi. Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance. *Neural Networks*, 21(2-3):427–436, 2008. doi: 10.1016/j.neunet.2007.12.031.

[31] 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. doi: 10.1016/j.neunet.2018.07.011.

[32] Alberto Fernandez, Salvador Garcia, Julián Luengo, Ester Bernado-Mansilla, and Francisco Herrera. Genetics-based machine learning for rule induction: State of the art, taxonomy, and comparative study. *IEEE Transactions on Evolutionary Computation*, 14(6):913–941, 2010. doi: 10.1109/tevc.2009.2039140.

[33] R. Anand, K.G. Mehrotra, C.K. Mohan, and S. Ranka. An improved algorithm for neural network classification of imbalanced training sets. *IEEE Transactions on Neural Networks*, 4(6):962–969, 1993. doi: 10.1109/72.286891.

[34] Haibo He and E.A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009. doi: 10.1109/tkde.2008.239.

[35] Will Badr. Having an imbalanced dataset? here is how you can fix it., 2019. URL `https://towardsdatascience.com/having-an-imbalanced-dataset-here-is-how-you-can-solve-it-1640568947eb`.

[36] Haixiang Guo, Yijing Li, Jennifer Shang, Gu Mingyun, Huang Yuanyue, and Gong Bing. Learning from class-imbalanced data: Review of methods and applications. *Expert Syst. Appl.*, 73:220–239, 2017.

[37] Gil Levi and Tal Hassner. Age and gender classification using convolutional neural networks. *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 34–42, 2015.

[38] Andrew Janowczyk and Anant Madabhushi. Deep learning for digital pathology image analysis: A comprehensive tutorial with selected use cases. In *Journal of pathology informatics*, 2016.

[39] Salman Hameed Khan, Munawar Hayat, Mohammed Bennamoun, Ferdous Ahmed Sohel, and Roberto Togneri. Cost-sensitive learning of deep feature representations from imbalanced data. *IEEE Transactions on Neural Networks and Learning Systems*, 29:3573–3587, 2015.

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

[41] D.J. Drown, T.M. Khoshgoftaar, and N. Seliya. Evolutionary sampling and software quality modeling of high-assurance systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 39(5):1097–1107, 2009. doi: 10.1109/tsmca.2009.2020804.

[42] Chris Drummond and Robert C. Holte. Class imbalance , and cost sensitivity : Why under-sampling beats oversampling. 2003.

[43] Alhussein Fawzi, Horst Samulowitz, Deepak S. Turaga, and Pascal Frossard. Adaptive data augmentation for image classification. *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3688–3692, 2016.

[44] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017. ISSN 0001-0782. doi: 10.1145/3065386. URL `http://doi.acm.org/10.1145/3065386`.

[45] Bharath Raj. Data augmentation | how to use deep learning when you have limited data — part 2, 2019. URL `https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c2`

[46] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The mnist database of handwritten digits.

[47] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew P. Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 105–114, 2016.

[48] Chuan Li and Michael Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks, 2016.

[49] Rui Huang, Shu Zhang, Tianyu Li, and Ran He. Beyond face rotation: Global and local perception gan for photorealistic and identity preserving frontal view synthesis. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2458–2467, 2017.

[50] Kevin McGuinness. Generative models and adversarial training (d3l4 2017 upc deep learning for computer vision), 2017.

[51] Chris Nicholson. A beginner's guide to generative adversarial networks (gans). URL `https://skymind.com/wiki/generative-adversarial-network-gan`.

[52] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[53] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. arxiv. 2017.

[54] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[55] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.

[56] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2017.

[57] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[58] 2019. URL `https://en.wikipedia.org/wiki/Wasserstein_metric`.

[59] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017.

[60] Xin Gao, Fang Deng, and Xianghu Yue. Data augmentation in fault diagnosis based on the wasserstein generative adversarial network with gradient penalty. *Neurocomputing*, 2019.

[61] Fei Wang, Zhanyao Zhang, Chun Liu, Yili Yu, Songling Pang, Neven Duić, Miadreza Shafie-Khah, and Joao PS Catalao. Generative adversarial networks and convolutional neural networks based weather classification model for day ahead short-term photovoltaic power forecasting. *Energy conversion and management*, 181:443–462, 2019.

[62] 2019. URL `https://keras.io/preprocessing/image/`.

[63] Francois Chollet. Building powerful image classification models using very little data, 2016. URL `https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html`.

[64] Adrian Rosebrock. Keras imagedatagenerator and data augmentation - pyimagesearch, 2019. URL `https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/`.

[65] Rakshith Vasudev. Understanding and calculating the number of parameters in convolution neural networks (cnns), 2019. URL `https://towardsdatascience.com/understanding-and-calculating-the-number-of-parameters-in-convolution-neural-ne`

[66] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? *arXiv preprint arXiv:1801.04406*, 2018.

[67] Giovanni Mariani, Florian Scheidegger, Roxana Istrate, Constantine Bekas, and A. Cristiano I. Malossi. Bagan: Data augmentation with balancing gan. *ArXiv*, abs/1803.09655, 2018.

[68] Zhou Wang, Alan C Bovik, Hamid R Sheikh, Eero P Simoncelli, et al. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[69] Xiao-Xiao Niu and Ching Y Suen. A novel hybrid cnn–svm classifier for recognizing handwritten digits. *Pattern Recognition*, 45(4):1318–1325, 2012.

[70] Xiaoxiao Niu. *Fusions of CNN and SVM Classifiers for Recognizing Handwritten Characters*. PhD thesis, Concordia University, 2011.

[71] Senzhang Wang, Zhoujun Li, Chunyang Liu, Xiaoming Zhang, and Haijun Zhang. Training data reduction to speed up svm training. *Applied intelligence*, 41(2): 405–420, 2014.

[72] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *Icdar*, volume 3, 2003.

[73] C Manliguez. Generalized confusion matrix for multiple classes. 2016.

[74] Nitesh V Chawla. Data mining for imbalanced datasets: An overview. In *Data mining and knowledge discovery handbook*, pages 875–886. Springer, 2009.

[75] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[76] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[77] Akarsh Singh. Deep learning tensorflow vs keras vs pytorch, 2017. URL `http://codeinpython.com/tutorials/deep-learning-tensorflow-keras-pytorch/`.

[78] Brooke Wenig and Jules S. Damji. A tale of three deep learning frameworks: Tensorflow, keras, pytorch with brooke wenig and jules damji, 2018.

[79] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL http://arxiv.org/abs/1605.02688.

[80] Dong Yu, Adam Eversole, Mike Seltzer, Kaisheng Yao, Zhiheng Huang, Brian Guenter, Oleksii Kuchaiev, Yu Zhang, Frank Seide, Huaming Wang, et al. An introduction to computational networks and the computational network toolkit. *Microsoft Technical Report MSR-TR-2014–112*, 2014.

[81] 2019. URL https://colab.research.google.com/notebooks/welcome.ipynb.

[82] 2019. URL https://research.zalando.com/.

[83] 2019. URL https://research.zalando.com/welcome/mission/research-projects/fashion-mnist/.

[84] 2019. URL https://medium.com/@rgrgrajat1/why-not-mnist-with-deep-learning-studio-6e7ec4e1450d.

[85] 2019. URL http://groups.csail.mit.edu/vision/TinyImages/.

[86] 2019. URL https://www.kaggle.com/benhamner/popular-datasets-over-time/code.

[87] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier GANs. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2642–2651, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/odena17a.html.

[88] Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3387–3395. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/6519-synthesizing-the-preferred-inputs-for-neurons-in-neural-networks-via-deep-g
pdf.

[89] Ali Borji. Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019. doi: 10.1016/j.cviu.2018.10.009.

[90] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2234–2242. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/6125-improved-techniques-for-training-gans.pdf.

[91] Yuancheng Ye, Lijuan Wang, Yue Wu, Yinpeng Chen, Yingli Tian, Zicheng Liu, and Zhengyou Zhang. Gan quality index (gqi) by gan-induced classifier, 2018. URL https://openreview.net/forum?id=S1CIev1vM.

[92] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier GANs. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2642–2651, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/odena17a.html.

[93] Victoria López, Alberto Fernández, Salvador García, Vasile Palade, and Francisco Herrera. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information Sciences*, 250: 113–141, 2013. doi: 10.1016/j.ins.2013.07.007.

[94] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[95] V. García, R. A. Mollineda, and J. S. Sánchez. Index of balanced accuracy: A performance measure for skewed class distributions. *Pattern Recognition and Image Analysis*, pages 441–448, 2009. doi: 10.1007/978-3-642-02172-5_57.

[96] William H. Jean and Billy P. Helms. Geometric mean approximations. *The Journal of Financial and Quantitative Analysis*, 18(3):287, 1983. doi: 10.2307/2330720.

[97] 2019. URL http://gim.unmc.edu/dxtests/roc3.htm.

[98] YANMIN SUN, ANDREW K. C. WONG, and MOHAMED S. KAMEL. Classification of imbalanced data: A review. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(04):687–719, 2009. doi: 10.1142/s0218001409007326.

[99] Misha Denil and Thomas Trappenberg. Overlap versus imbalance. *Advances in Artificial Intelligence*, pages 220–231, 2010. doi: 10.1007/978-3-642-13059-5_22.

[100] V. García, R. A. Mollineda, and J. S. Sánchez. On the k-nn performance in a challenging scenario of imbalance and overlapping. *Pattern Analysis and Applications*, 11(3-4):269–280, 2007. doi: 10.1007/s10044-007-0087-5.

[101] C. E. Brodley and M. A. Friedl. Identifying mislabeled training data. 2011. doi: 10.1613/jair.606.

[102] Chris Seiffert, Taghi M. Khoshgoftaar, Jason Van Hulse, and Andres Folleco. An empirical study of the classification performance of learners on imbalanced and noisy software quality data. *Information Sciences*, 259:571–595, 2014. doi: 10.1016/j.ins.2010.12.016.

[103] Gary M. Weiss. Mining with rarity: A unifying framework. *SIGKDD Explor. Newsl.*, 6 (1):7–19, June 2004. ISSN 1931-0145. doi: 10.1145/1007730.1007734. URL `http://doi.acm.org/10.1145/1007730.1007734`.

[104] Jose G. Moreno-Torres, Xavier Llorà, David E. Goldberg, and Rohit Bhargava. Repairing fractures between data using genetic programming-based feature extraction: A case study in cancer diagnosis. *Information Sciences*, 222:805–823, 2013. doi: 10.1016/j.ins.2010.09.018.

[105] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90 (2):227–244, 2000. doi: 10.1016/s0378-3758(00)00115-4.

[106] Krystyna Napierała, Jerzy Stefanowski, and Szymon Wilk. Learning from imbalanced data in presence of noisy and borderline examples. *Rough Sets and Current Trends in Computing*, pages 158–167, 2010. doi: 10.1007/978-3-642-13529-3_18.

[107] Mike Wasikowski and Xue-wen Chen. Combating the small sample class imbalance problem using feature selection. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1388–1400, 2010. doi: 10.1109/tkde.2009.187.

[108] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[109] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 2012. ISSN 0893-6080. doi: 10.1016/j.neunet.2012.02.016. URL `http://www.sciencedirect.com/science/article/pii/S0893608012000457`.

[110] J Alcalá-Fdez, A Fernandez, J Luengo, J Derrac, S Garcia, L Sánchez, and F Herrera. Keel data-mining software tool: Data set repository, integration of algorithms and

experimental analysis framework. . *Journal of Multiple-Valued Logic and Soft Computing*, 17:255–287, 2011.

[111] 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.

[112] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. Borderline-smote: A new over-sampling method in imbalanced data sets learning. *Lecture Notes in Computer Science*, pages 878–887, 2005. doi: 10.1007/11538059_91.

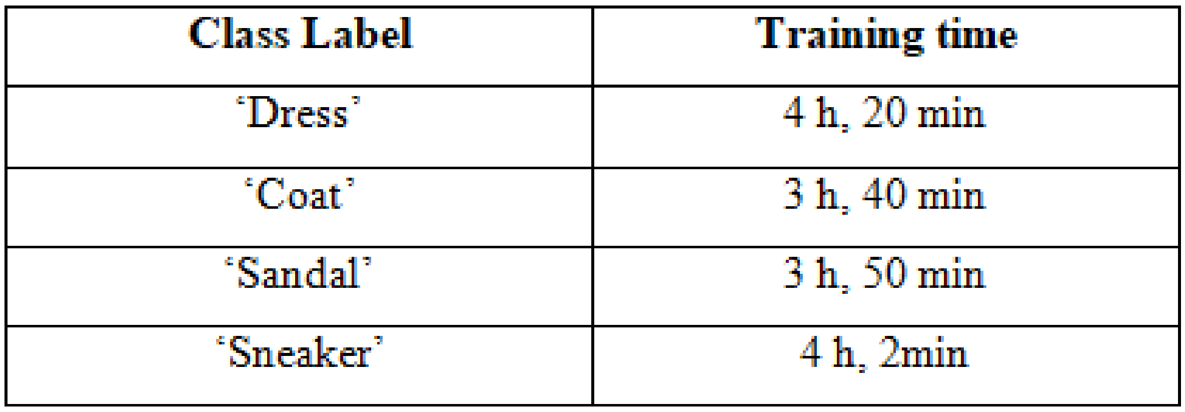
[113] Chumphol Bunkhumpornpat, Krung Sinapiromsaran, and Chidchanok Lursinsap. Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem. *Advances in Knowledge Discovery and Data Mining*, pages 475–482, 2009. doi: 10.1007/978-3-642-01307-2_43.

Appendix A

The time taken to train the WGAN-GP on the various imbalanced classes of the two datasets is recorded here, to give an estimation of the computational complexity and cost involved. The technical set-up has been described in Chapter 4.

WGAN-GP Train Durations for FMNIST

The four imbalanced classes in this dataset are 'Sandal', 'Dress', 'Sneaker' and 'Coat', mentioned in order of ascending imbalance (i.e. the class 'Coat' contains the least quantity of samples among these four classes). The train times for each of these classes is shown in Table A1.

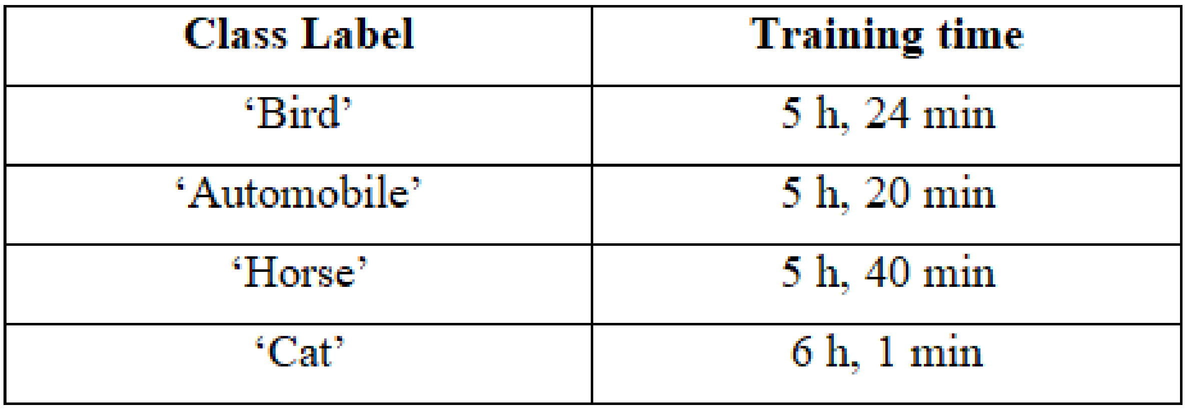


Class Label	Training time
'Dress'	4 h, 20 min
'Coat'	3 h, 40 min
'Sandal'	3 h, 50 min
'Sneaker'	4 h, 2min

Table A1: Train Times of the Minority Classes of Imbalanced-FMNIST on WGAN-GP

WGAN-GP Train Durations for CIFAR-10

The four imbalanced classes in this dataset are 'automobile', 'bird', 'cat' and 'horse', mentioned in order of ascending imbalance (i.e. the class 'horse' contains the least number of samples among these four classes). The train times for each of these classes is shown in Table A2.



Class Label	Training time
‘Bird’	5 h, 24 min
‘Automobile’	5 h, 20 min
‘Horse’	5 h, 40 min
‘Cat’	6 h, 1 min

Table A2: Train Times of the Minority Classes of Imbalanced CIFAR-10 on WGAN-GP

Appendix B

The codes and scripts implemented for the purpose of this study, along with all relevant datasets, results, saved models and compile and deployment instructions have been made available at the following link:

https://drive.google.com/drive/folders/1oGHEXht-
GGm1keeriqP4HN9ZBIB7lQ1V?usp=sharing