KU LEUVEN

FACULTY OF SCIENCE

Statistical measures for evaluating generative models with an application to fraud detection

Blazej Herbert MAKOSA

Promotor: Prof. B. De Moor
Supervisor: *H. De Meulemeester*

Thesis presented in
fulfillment of the requirements
for the degree of Master of Science
in Statistics and Data Science

Academic year 2020-2021

Abbreviations

CNN	Convolutional Neural Network
DBM	Deep Boltzmann Machine
DCGAN	Deep Convolutional Generative Adversarial Network
FID	Frechet Inception Distance
GAN	Generative Adversarial Network
IS	Inception Score
KMMD	Kernel Maximum Mean Discrepancy
LSGAN	Least Squares Generative Adversarial Network
MMD	Maximum Mean Discrepancy
NN	Neural Network
ReLU	Rectified Linear Unit
RBM	Restricted Boltzmann Machine
WGAN	Wasserstein Generative Adversarial Network

Contents

Chapter 1

Introduction

In this thesis we investigate two main topics. The first topic is the use of generative adversarial network model evaluation measures for fraud, or outlier detection. These evaluation measures determine the distance between samples of observations, usually a sample from the target population and a sample of generated observation. The second topic is the possibility, or viability, of using the generative adversarial network's discriminator as a feature space, in which the aforementioned distance measures could be calculated in.

The setting for the work involves the idea of applying the two topics to the problem of unsupervised fraud detection. More specifically, the environment in which the behaviours are explored supposes that there are no examples of fraudulent observations available during training. This means that the problem is an outlier detection problem. Where the distance measures are used to determine a distance for new observations, and this distance is then used to determine whether the new observations are outliers or not. If they are, they are considered fraudulent samples or observations.

GAN models are trained on the available, non-fraudulent, data in order to obtain a feature space from the GAN discriminator. The distances between the known non-fraudulent observations and unknown fraudulent and non-fraudulent observations are calculated in the feature space created by the GAN discriminator network. There are three main experiments. Each focusing on different factors that may affect the behaviours of the distances that are calculated. These factors include: the type of GAN model used, the inclusion of dimensionality reduction techniques, and the proportion of fraudulent observations in the sample.

Chapter 2

Literature Review

2.1 Generative models

Generative models, as the name implies, are models that generate samples $x_1, ..., x_n$ from a target distribution $p(\boldsymbol{x})$ and can be extended to produce samples of label y for a target distribution $p(\boldsymbol{x}|y)$.

Generative models can be divided into two groups: prescribed (explicit) and implicit models [12]. We can see a visual illustration of a family tree of different popular generative modes in Figure 2.1.

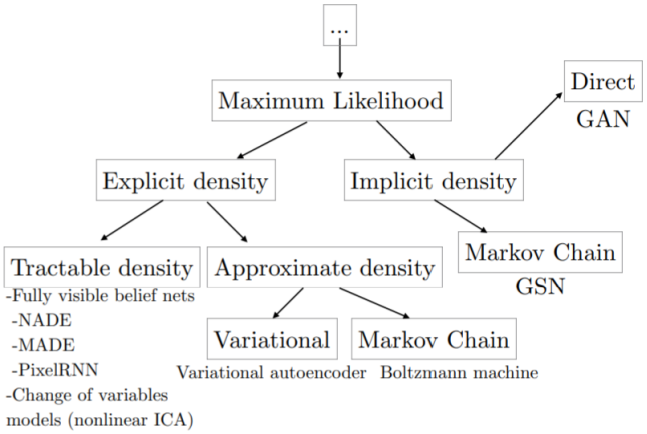


Figure 2.1: Tree of generative models. Taken from [17]

2.1.1 Explicit density models

Explicit density models are such that provide an explicit parametric definition of the distribution of the target distribution x and specify a log-likelihood function for this distribution. These models then aim to maximise the likelihood, or to maximise an approximation of the likelihood. For some, the density function is tractable and can be optimised directly, whereas for other models the density is intractable but can be derived and an approximation can be optimised.

2.1.1.1 PixelRNN/PixelCNN

PixelRNN [38] and PixelCNN [39] models generate images based on a pixel-by-pixel approach. This allows for the likelihood function to be explicitly computed. Each image, $\mathbf{x}$ is assigned a probability $p(\mathbf{x})$ and is formed of $n \times n$ pixels. The pixels are treated as a one-dimensional sequence of values $x_1, \ldots, x_{n^2}$, where they are arranged row after row. With this structure, the joint distribution $p(\mathbf{x})$, can be estimated using the product of the conditional distributions over the pixels.

$$p(\mathbf{x}) = \prod_{n=1}^{n^2} p(x_i | x_1, \ldots, x_{i-1}) \tag{2.1}$$

With this approach, the pixel values are considered discrete variables. If the image is in colour, then each pixel consists of 3 discrete variables corresponding to each colour of the RGB spectrum. In that case each pixel can be written as the joint distribution of these three variables.

For a PixelCNN model, the dependency of a pixel on the previous pixel values is modelled by a convolutional neural network. Whereas, in a PixelRNN model this is done using a recurrent neural network model.

When generating new images using these models, the pixels are generated individually starting from a corner. The subsequent pixels are dependent on the pixel values that have been previously generated.

2.1.1.2 Variational Auto Encoders

Variational autoencoder models have a similar structure to autoencoder models. Autoencoder models employ an unsupervised approach for learning a feature representation of unlabelled data [13]. An autoencoder is a feedforward neural network that can be separated into two main parts: the encoder and the decoder networks. The encoder network takes as input the training data and outputs the feature representation of this data. The decoder network takes as input the feature representation of the data and provides as output a reconstruction of the input data based on the feature space representation.

Variational autoencoder models differ from standard autoencoder models by the additional constraints imposed on the encoded representations that are being learned by the model. In a variational autoencoder model, the encoder learns a latent variable representation for its input data. This latent space usually consists of two parameters, the mean and the log variance. Thus, the latent distribution is, usually, a normal distribution. In order to generate data, a sample is drawn from the latent variable distribution. The decoder network maps the latent space points to the original input data. This way each training observation corresponds to an observation in the latent space distribution. A new sample from the latent space distribution should correspond to a new sample in the original space, if the model has been trained successfully. Observations that are close to each other in the latent space will be close to each other in the original space.

We can describe in probabilistic terms the way in which a VAE model approximates the density of the target distribution. Denote the data likelihood as $p_\theta(x)$, where θ denotes the parameters of the decoder network. Assume that we have selected a simple gaussian prior, $p_\theta(z)$ and have trained a decoder network that represents the likelihood of the data given the prior, $p_\theta(x|z)$. Then we obtain an intractable expression for the data likelihood.

$$p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz \tag{2.2}$$

The posterior density $p_\theta(x|z)$ is also intractable. However, it can be approximated using the encoder network, $q_\Theta(z|x)$. Using this approximation, we are able to obtain a tractable lower bound on the log-likelihood of the data. Thus, we have an explicit density generative model.

An important element that makes the VAE model work is the reparameterisation trick by Kingma [27]. The reparametrisation trick allows for the Monte Carlo estimate of the expectation to be differentiable during the training process. Without it, there is no guarantee that the derivative estimate will converge correctly.

2.1.2 Implicit density models

Implicit density models do not rely on a likelihood function or a likelihood function approximation. Implicit density models define a stochastic procedure that directly generates the data.

2.1.2.1 Generative Adversarial Network

The Generative Adversarial Network model [17] is a type of implicit density generative model. The model transforms the input, a sample from a simple distribution, to an observation belonging to the target distribution. There are no likelihood functions, prior distributions, or conditional distributions that are required to be provided or calculated in order for the model to function. The model is a neural network, within which there is no tractable likelihood.

2.2 Generative Adversarial Networks

In order to understand Generative Adversarial Networks and their evaluation measures one must first understand the concepts of: neural networks, convolutional neural networks, and adversarial training. After covering these ideas, we will be able to cover different GAN architectures and explore GAN evaluation measures.

2.2.1 Neural Networks

Neural Network models all stem from the basic idea of mimicking the behaviour of a human brain in a mathematical manner. Considering the immense cognitive abilities of human brains, it makes sense to suppose that a mathematical model that could replicate or mimic the working of a human brain could prove to be a powerful tool.

Despite the impression that one may have that neural networks are novel, it is not the case. In 1943, McCulloch [34] proposed the precursor to what is now known as the perceptron model. The perceptron model was later used to create more complex models called multilayer perceptron models.

Since the turn of the century there has been a lot of development in the area of neural networks. This is due to the increasing computational capabilities of modern computers.

Previously the barrier towards development within the field was the lack of appropriate computing capabilities.

2.2.1.1 Perceptron

The perceptron model [43] can be compared to a single neuron in our brains. Suppose we have an input vector X of dimension n and an output Y which is a scalar. For each element, x_i in the input vector, there exists an associated weight element w_i. We can store these weights in the vector W.

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

In the model, these input elements and the weight elements are multiplied with each other. Also, a bias term, b is added to the resulting value. The final result of this operation is the scalar $Z = z$

$$Z = W^T X + b = w_1 x_1 + w_2 x_2 + ... + w_n x_n + b \tag{2.3}$$

The final step within the perceptron model is to put the scalar Z through an activation function $f()$. There are many different activation functions that one can choose from. The choice of the activation function depends on the nature of the output variable.

$$f(Z) = Y_{pred}, \text{ where } f(.) \text{ is some activation function.} \tag{2.4}$$

The visual representation of this model in Figure 2.2, shows the parallel structure between this model and the neurons in our brain.

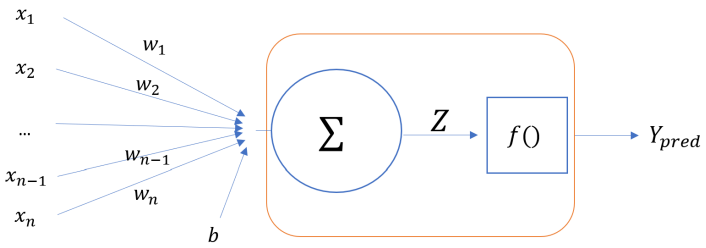


Figure 2.2: Illustration of the perceptron model

2.2.1.2 Multi Layer Perceptron

Multi Layer Perceptron (MLP) [44] networks employ perceptron models as a building blocks. The perceptrons are arranged following a layered structure. Excluding the first and final layers, each perceptron in a layer is connected to every perceptron in the previous layer, as well as each perceptron in the following layer. This arrangement is called a fully connected network. The nodes in the first layer refer to the values in the input vector,

with the number of nodes being equal to the length of the input vector. The perceptrons in the final layer each provide an output the same way as in the perceptron model. In Figure 2.3 we see an illustration of the fully connected multilayer network model, with one output. We can see the layers within the network and the connections between the neurons in each layer.

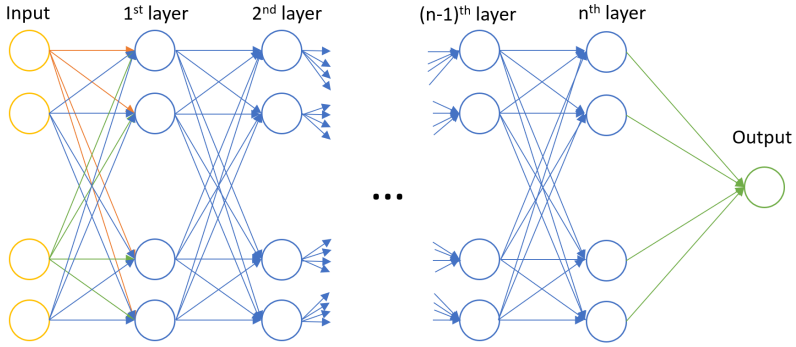


Figure 2.3: Illustration of the MLP model

This complex structure allows the MLP network to model far more complex functions than a single perceptron. Considering that the number of outputs of an MLP network corresponds to the number of neurons in the final layer, it can model functions that require several outputs. Something the perceptron model is not capable of.

2.2.1.3 Activation functions

Within a multilayer network, the neurons within each layer use the same activation function. However, different layers may use different activation functions. The choice of activation function depends on the type of data and model that is used. Generative Adversarial networks tend to use activations function similar to those used within computer vision. These include the ReLU and Leaky ReLU functions, as well as the tanh and sigmoid functions.

ReLU stands for Rectified Linear Unit. The ReLU activation function is a piecewise linear activation function that is equal to $f(x) = x$ from zero onwards, at and below zero the function is equal to $f(x) = 0$. The Leaky ReLU activation function has the same behaviour for values of x above zero, however at and below zero the function is equal to $f(x) = 0.01x$. Thus, it allows for negative output values. The ReLU family of activation functions is very commonly used in computer vision. Some of the advantageous properties include sparse activation, scale invariance, and efficient computation. The sparse activation is cause by the fact that large negative values lead either to zero or very small output values.

2.2.2 Convolutional Neural Networks

Convolutional neural networks [31] are multilayer neural networks that are not fully connected, instead they are locally connected. This local connectivity is implemented through convolutional layers. Within these layers, the neurons are not connected to every neuron in the subsequent layer but instead are only connected to a small subset of neurons that are located nearby. This property of local connectivity has been proven to be beneficial

when working with data such as text and images. It is logical that the most important features within images or texts will be those surrounding the pixel or word in question. One of the first implementations of convolutional neural networks was in LeNet-5 by Yann LeCun [32].

Convolutions are mathematical transformations on two functions u and v that generate a third function. This is illustrated in the equation below.

$$(u * v)(x) = \int u(x-t)v(t)dt \tag{2.5}$$

Convolutions satisfy several algebraic properties that make them useful for NN applications. These properties include commutativity, associativity, associativity with scalar multiplication, and distributivity.

In CNNs, each convolutional layer contains a specific kernel operation that determines the type of convolution that is performed in said layer. Depending on the type of convolution, the kernel operation can be matrix multiplication, tensor multiplication, or a rule-based operation. The size of the kernel determines the input of the kernel operation, and the stride affects the size of the output layer.

Pooling layers are used to reduce the dimension of the layers in the network. The input layer is split into equal sized regions that each then correspond to single neurons in the subsequent layer. There are different rules for determining the value that is fed to the neuron in the subsequent layer. Max pooling is a commonly used type of decision rule, where the maximum value within each region is selected as the input to the corresponding neuron in the following layer. Other decision rules include average pooling where the average of the values within the region is taken, and L2-Norm pooling where the L2-Norm of the values in the region is taken. Below an illustration of max pooling is shown.

By applying 2-by-2, stride 2, max-pooling: $\left[\begin{array}{cc|cc} 1 & 2 & 1 & 4 \\ 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 2 & 6 \\ 1 & 0 & 1 & 3 \end{array}\right]$ becomes $\left[\begin{array}{c|c} 2 & 4 \\ \hline 1 & 6 \end{array}\right]$

2.2.3 Training Neural Networks

As with any other type of model, neural network models have to be trained in order to output the desired results. During this training process, the weight and bias parameters of the network will be updated. The training process explained below is a supervised learning process, thus it requires that every training observation has a corresponding response value. The process itself can be split into several stages: set-up, forward pass, loss calculation, and backpropagation.

2.2.3.1 Set-up

Before the training can begin, all the necessary elements have to be set-up. First, the data has to be prepared. Usually, data requires some kind of pre-processing. For example, image data is usually standardised. The pixel values range from 0 to 255, and are usually transform to fit in the range $[0, 1]$. For other data types, other pre-processing steps may be necessary.

Once the data is pre-processed, usually it is split into training, validation, and test sets. The training set is used for training where the parameters are updated. The validation set is used to evaluate the training performance on a set that does not affect the model parameters. The performance on the validation set can help identify problem such as overfitting. Finally, the test set is used to assess the network performance once training is completed.

During the training process, a loss function and an optimiser are used. These have to be chosen before the commencement of training. Finally, the network parameters are initialised. Usually, they are assigned some randomly generated values.

2.2.3.2 Forward pass

During the forward pass, the training observations are fed through the network and their predicted outputs are stored. This means that the training data is put into the network and fed through all the layers, in which the necessary calculations are performed using the existing weights. During the first forward pass the initialised weights from the set-up are used.

2.2.3.3 Loss function evaluation

Once the training observations have completed the forward pass, a set of calculated outputs is available. These are the predicted response values using the current model. These predicted values are compared to the true response values via the loss function. There are many different loss functions that are used depending on the nature of the response variable. The Mean Square Loss function is often used for continuous responses. Other continuous loss functions include the Mean Absolute Loss function. For categorical responses, the Cross-Entropy loss function is commonly used both for binary and multi-class problems. Other options include the sparse Cross-entropy loss, and the DICE loss functions. Using the chosen loss function and both the predicted responses and true responses, the loss value of the model is calculated.

2.2.3.4 Backpropagation

The next step in the training process is to perform backpropagation. During backpropagation, the optimiser is used to calculated how the parameters in the network should be modified in order to decrease the loss value of the model. Optimisers are versions of stochastic gradient descend that are modified in order to improve their performance respective to classical stochastic gradient descend. The benefits of the modifications include shorted training times, lower likelihood of being stuck in local minima. The most popular optimisers are the Adam optimiser [26] and the RMSprop optimiser [21].

Stochastic gradient descend is based on standard gradient descend which is an iterative optimization algorithm where the minimum of a function is obtained by using its differential. The differential provides the gradient, the negative of which is the direction in which the local minimum is located. The length of the movement taken in that direction is the learning rate, lr. Within a neural network the goal is to find the direction in which to change each parameter, weight or bias, that we want to modify. Thus, we take the derivative of the loss function with respect to each parameter separately. In multilayer

networks, the derivatives are calculated using the chain rule of differentiation, connecting the layers together.

$$\frac{\partial J}{\partial w_i} = [\sum (y_i - f(z_i))](-x_i) \tag{2.6}$$

The derivative provides the gradient in the direction of the maximum of the function, thus in order to obtain the direction towards the minimum the sign of the direction has to be changed.

$$w_i - lr \cdot \frac{\partial J}{\partial w_i} \rightarrow w_i^{updated} \tag{2.7}$$

The process of calculating the loss for the entire training dataset and updating the weights is called an epoch. Epochs are performed until a certain stopping criterion is met, usually the most common stopping criterion is the lack of change in the loss function value between two subsequent epochs. This indicates that it is not possible to descent along the gradient and thus either the global optimum is reached or at least some local optimum is reached.

Most current neural network implementations use batch training methods. The training data is split into many batches, which are samples of a certain size. The derivatives are calculated, and network weights updated after each batch of training data. Considering that most training datasets are very large, waiting to update the network until the whole training dataset has been fed through the network would results in long training times. With batch training, the training speed is increased as the weights are updated more frequently and the calculation necessary to perform the updates are easier.

2.2.4 Adversarial Training

Generative Adversarial Networks use an adversarial training method. GAN models consist of two neural networks that compete against each other. The adversarial training allows for the GAN model to learn the implicit distribution of the training data. Each of the individual neural networks is trained in a similar way as a standard neural network would be trained. The defining characteristic of a GAN model is the way in which the two networks interact.

2.2.4.1 NN Implementation

For the neural network implementation, we need two neural networks: the generator and the discriminator. The generator takes as input random noise and outputs objects of the same shape as those in the target distribution. The discriminator takes as input objects of the shape of those in the target distribution and outputs a value between 0 and 1. These output values refer to two possible classes that an observation may be assigned.

The first step in the training process is to train the discriminator. In order to do this, new observations are generated by the generator and an equal amount of observations are taken from the training data. Observations generated by the generator are assigned to class 0. The training data observations are assigned to class 1. Class 0 is often called

fake, and class 1 is often called true. The discriminator is trained to predict the classes of these observations correctly.

After the discriminator is trained, the generator is trained. To train the generator, both the generator and the discriminator are required. During the training process the two networks are connected, i.e. the output of the generator goes straight into the discriminator. The weights of the discriminator are fixed, meaning that they are not subject to backpropagation. The generator is trained so that inputs are predicted to belong to class 1 by the discriminator. Thus the generator is trained to create observations that are predicted to be true by the discriminator.

The training focus alternates between the discriminator and the generator until the process is finished. In an ideal scenario, the training focus would switch from the discriminator, once the discriminator is able to predict all the observations correctly. And it would switch from the generator once all the generated observations would be predicted to be true by the discriminator. Usually, in practice, for each epoch, the discriminator is trained once and then the generator is trained once. The theoretical stopping condition would be the point at which the discriminator accuracy would be 50%, so as good as flipping a coin. This would imply that the generated samples are indistinguishable from the real ones. In practice, the discriminator accuracy does not have to converge to 50%. Therefore, the model is usually deemed to be trained by visually inspecting the generated samples, or by using a GAN evaluation measure.

2.2.5 Evolution of Generative Adversarial Networks

Since their introduction in 2014, many new GAN based models have been developed. Each subsequent model attempts to surpass the performance of the previously available models. There have been different approaches as to where the improvement originates from. Currently there are many different viable GAN model architectures most of which can achieve good results. We will discuss a selected few of these models.

2.2.5.1 Generative Adversarial Nets

We start with the first GAN model proposed by Goodfellow [16] in 2014. The GAN model proposed in the paper consists of two multilayer perceptrons - fully connected networks. One is the generator - G - and the other is the discriminator - D. The generator takes as input noise variables z that have some defined prior $p_z(z)$. The generator learns a mapping $G(z|theta_g)$ from the input noise to the data x with parameters θ_g in the neural network. The outputs of the generator belong to the generator distribution p_g. The discriminator gives as output a single scalar. This output $D(x)$ represents the probability that x belongs to the real data rather than the generator distribution p_g.

The discriminator D is trained as to maximise the probability of correctly labelling the training examples, those that belong to the real data and those that belong to p_g. The generator G is simultaneously trained to minimise $log(1 - D(G(z)))$. This results in D and G engaging in a two-player minimax game with the following value function $V(G, D)$.

$$\min_G \max_D \quad V_{GAN}(D,G) = \mathbb{E}_{x\sim p_{data(x)}}[log(D(x))] + \mathbb{E}_{z\sim p_z(z)}[log(1 - D(G(z)))] \tag{2.8}$$

Above is the objective function that would be used as the loss function during the training process of the GAN model. In practice the two networks are trained separately, resulting in two loss functions. The separate discriminator and generator objective functions are shown below.

$$V_D(D,G) = \mathbb{E}_{x\sim p_{data}}[log(D(x))] + \mathbb{E}_{z\sim p_z}[log(1-D(z))] \tag{2.9}$$
$$V_G(D,G) = -\mathbb{E}_{z\sim p_z}[log(D(z))] \tag{2.10}$$

2.2.5.2 Deep Convolutional GAN

The first popular implementation of the generative adversarial neural network was presented by Radford et al. [42] in 2015. This model was named DCGAN - Deep Convolutional Generative Adversarial Network. Compared to the model proposed by Goodfellow, this model uses CNN rather than MLP architectures in its networks. Also, the original proposal included maxout activation functions, whereas the DCGAN model uses ReLU activation functions. Finally, the DCGAN model uses batch normalisation. An illustration of the model architecture is shown in Figure 2.4. This architecture proved to be stable to train and capable of generating good quality observations. These qualities made it popular.

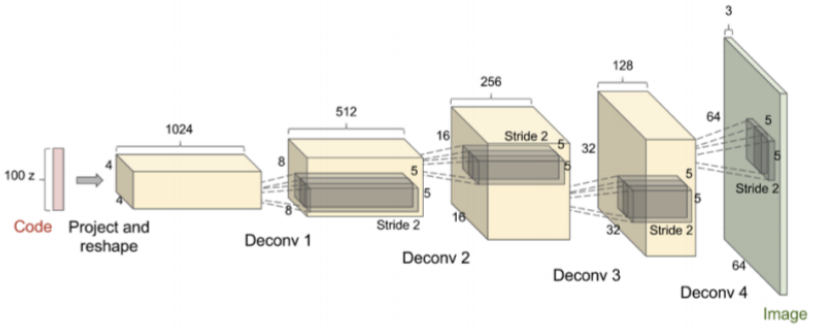


Figure 2.4: DCGAN generator architecture. Taken from [42]

2.2.5.3 Wasserstein GAN

The Wasserstein GAN (WGAN) model was proposed by Arjovsky et al. [3] in 2017. In this paper several different measures for calculating the distance between two distributions are analysed. These are the Total Variation distance, Kullback-Leiber divergence, Jensen-Shannon divergence, and Earth-Mover or Wasserstein-1 distance. It demonstrates that the Wasserstein-1 distance is a continuous loss function under mild assumptions and is the most sensible choice for learning distributions supported by low dimensional manifolds. Wasserstein GAN models claim to have two main benefits over other models. The first benefit is that the loss metric correlates with the convergence of the generator network, and with the quality of the generated samples. The second claim is that WGAN models offer improved training stability.

The WGAN model uses this Wasserstein-1 distance during the training process instead of the original GAN objective function $V_{GAN}(D,G)$. Of course, in practice the the discriminator and generator objective functions are separate. They can be seen below.

$$V_D(D,G) = \mathbb{E}_{x\sim p_{data}}[D(x)] - \mathbb{E}_{z\sim p_z}[D(G(z))] \tag{2.11}$$
$$V_G(D,G) = -V_D(D,G) \tag{2.12}$$

An improvement on the Wasserstein GAN is the Wasserstein GAN with gradient penalty (GP-WGAN) [18]. For this model, the the discriminator loss is modified as to include a gradient penalty. This gradient penalty aims to better enforce the Lipshitz constraint. GP-WGAN models claim to provide modelling performance and training stability compared to the WGAN model. The modified objective functions for the discriminator and generator are shown below.

$$V_D(D,G) = \mathbb{E}_{x\sim p_{data}}[D(x)] - \mathbb{E}_{z\sim p_z}[D(G(z))] + \lambda\,\mathbb{E}_{x\sim p_x}[(||\nabla_x D(x)||_2 - 1)^2] \tag{2.13}$$
$$V_G(D,G) = -V_D(D,G) \tag{2.14}$$

2.2.5.4 Least Squares GAN

Least Squares Generative Adversarial Networks (LSGAN) were proposed by Mao et al. [33] in 2017. The goal of the LSGAN model was to attempt to minimise the chance of the vanishing gradient problem occurring. This is done by replacing the minmax objective function in the original GAN architecture by a least squares loss objective function. We can build LSGAN objective functions using the original GAN functions as a basis. For this, we need to assign labels to the coding scheme of the discriminator. We can take a as the label for fake data, and b as the label for real data. We also denote C as the value that G wants D to believe for fake data. Then we can show the objective functions for the LSGAN model.

$$\min_D V_{LSGAN}(D) = \frac{1}{2}\,\mathbb{E}_{x\sim p_{data(x)}}[(D(x)-b)^2] + \frac{1}{2}\,\mathbb{E}_{z\sim p_z(z)}[(D(G(z))-a)^2] \tag{2.15}$$
$$\min_G V_{LSGAN}(G) = \frac{1}{2}\,\mathbb{E}_{z\sim p_z(z)}[(D(G(z))-c)^2] \tag{2.16}$$

LSGAN models claim to be able to generate samples that are closer to the target data, due to being able to move fake samples closer towards the decision boundary of the discriminator.

2.2.6 Issues with Generative Adversarial Networks

GAN based models are being used to achieve impressive results in many different subject areas. However, there are several different problems that are common among GAN based models. Most of the problems encountered when working with GAN models occur during the training process.

2.2.6.1 Vanishing Gradient

The vanishing gradient problem is common to many NN models. This problem occurs during the backpropagation step of the training process of NNs. During back-propagation,

the connection weights in the network are updated proportionally to the partial derivative of the loss function with respect to the current weights. In some situations, this partial derivative (gradient) may become very small and thus the changes in the weights may be non-existent.

In the specific case of GANs, as proven by Arjosky and Bottou [2] as well as Sinn and Rawat [48], the vanishing gradient is likely to occur when the generator is fixed and the discriminator is trained until optimality. In such a situation, the gradients of the discriminator may vanish, and further updates of the generator may become impossible. This is because in such a situation, the Kullback-Liebler divergences will be maxed out, and it is not possible to minimize them by gradient descent. More specifically, Arjovky and Bottou establish that this is almost surely going to happen if the dimension of the random input is smaller than the dimension of the target data, and the generator is parametrized by a standard neural network.

2.2.6.2 Mode Collapse

The initial GAN paper by Goodfellow [16] suggests that given 'sufficiently large' deep nets, sample size, and computation time, GANs do in fact learn the target distribution. Since then, doubt has been raised whether this statement is indeed correct. Using the Birthday Paradox for testing the diversity of images in a distribution, Arora et al. [4] suggest that current GAN approaches, specifically ones that produce images of higher visual quality, fall significantly short of learning the target distribution. The support size of the generated distribution is rather low, thus mode collapse occurs. Supposing that one has a target dataset containing many different subgroups, for example species of animals. If a model trained on this target dataset would suffer from mode collapse, one would expect that a dataset containing generated observations would not include some of the subgroups. Thus using our example, some of the species would not appear in the generated observations.

2.2.6.3 Possible Solutions

Since most of the problems are encountered during the training process, attempts to solve or minimise these problems involve modifying the training process. Different new GAN models attempt to solve some of these problems. For example, the paper in which WGAN models are introduced by Arjovsky et al. [3], claims that in their experiment there was no evidence of mode collapse for the WGAN algorithm. The LSGAN paper by Mao et al. [33] claims that LSGAN models generate more gradients allowing the generator to update more than other models, this aims to reduce the vanishing gradient problem.

2.3 Evaluation Measures

There are many different evaluation measures that can are used to evaluate the generative performance of GAN models. Many of these measures are created with image data in mind. The measures can be divided into two categories: quantitative and qualitative [7].

There are several capabilities that evaluation measures are desired to possess. These capabilities are based on the desired properties of a good GAN model. A good evaluation measure should favour models whose generated samples are diverse. It should not be

affected by image transformations, so the evaluation measure score should not be affected by small shifts or rotations. It should have the ability to distinguish between generated samples and real ones, thus exhibit discriminability. And finally, it should agree with the human evaluation of the model performance.

In the paper by Borji [7], many evaluation measures are listed and discussed. The different measures focus on different aspects of the desired GAN outputs. Some measures only consider images generated by the model and assess the performance by determining the quality and diversity of said images. Whereas other measures attempt to compare the distribution of the training images and those generated by the model. The most commonly used measure is the Inception Score which belongs to the former category. We will discuss this measure because of its popularity. However, we will then focus on measures that belong to the latter category. As we are interested in measures that provide an estimate of the distance or difference between the distributions of the generated and real datasets. As we want to apply these evaluation measures to a fraud detection situation.

2.3.1 Inception Score

The Inception Score evaluation method was proposed by Salimans et al. [46] in 2016. It uses the Inception Net [49], a pre-trained image classifier, to get a conditional label distribution $p(y|x)$ for each generated image. The focus of this metric in on two attributes of the generated observations, their classifiability and diversity. So, we can expect a good score if the generated observations are classified to labels with a high accuracy score/probability scores and if the predicted labels are varied and evenly distributed.

More specifically, the Inception Score is calculated as follows:

$$\exp(\mathbb{E}_x[KL(p(y|x)||p(y))]) = exp(H(Y) - \mathbb{E}_x[H(y|x)]) \tag{2.17}$$

Where $p(y|x)$ is the conditional label distribution for image x estimated using the pre-trained Inception model, and $p(y)$ is the marginal distribution $p(y) \approx \frac{1}{N}\sum_{n=1}^{N} p(y|x_n = G(Z_n))$. And, $H(x)$ represents the entropy of variable x.

There are some problems with the Inception Score as shown by Barrat and Sharma [6]. They point out that the Inception Score is sensitive to small changes in the network weights of the Inception Network. Although these changes in weights have little effect on the classification performance of the network, the Inception Score can be strongly affected. They also show that the mean Inception Score can change depending on the library in which the Inception Network is implemented in (Tensorflow [1], Pytorch [40], etc.). They also point out that applying the Inception Score on models trained on datasets other than ImageNet [11] can give misleading results.

Overall, in theory the Inception Score should perform well for measuring the diversity of the generated images and could be useful for identifying mode collapse. As mode collapse results in low variety in the image subjects, which would result in a low Inception Score result. However, it does not provide the ability to compare the distributions of two samples directly. When using it to assess a generative model, only the generative ability of said model is evaluated. The distribution of the generated images is not compared to the target distribution [53].

2.3.2 Wasserstein Distance

The Wasserstein distance [51], also known as the Wasserstein-1 distance or the Earth Movers Distance, is used to calculate the distance between two probability density functions. To calculate the Wasserstein distance, the minimum amount of energy required to transform the densities from one to the other is calculated. This idea is often illustrated by imagining the moving of sand from a heap to a pit. Where the shapes of the heap and the pit correspond to the shapes of the probability density functions. The goal is to find the way of moving the sand from the heap into the pit which uses the least energy possible, i.e finding the optimal way to transport the sand. This problem corresponds to a variation of the optimal transport problem.

The Wasserstein distance between μ and υ can take the form of the equation below, where $c(x, y)$ is the cost of transporting one unit of mass from x to y.

$$C(\mu, \upsilon) = inf_{\pi in \Pi(\mu,\upsilon)} \int c(x, y) d\pi(x, y) \tag{2.18}$$

The Wasserstein distance is estimated by the Wasserstein critic in a WGAN model. The Wasserstein critic is a neural network with clipped weights. There are also other variations of the Wasserstein distance such as the Sliced Wasserstein distance (SWD) [28]. These variations usually aim to reduce the computational cost required in comparison to that of the standard Wasserstein distance. As shown in by Xu et al. [53], the Wasserstein distance has high computational complexity making it less preferable to other measures despite performing well in different tests.

2.3.3 Kernel Maximum Mean Discrepancy

The Maximum Mean Discrepancy measure calculates the dissimilarity between two probability distributions P_r and $P - g$ using independently drawn samples from each of the distributions. Thus, MMD can be seen as a type of two-sample test.

Kernel MMD measures the square MMD between two distributions P_r and P_g using a fixed characteristic kernel function k. This kernel function can, for example, be a Gaussian kernel.

$$M_k(P_r, P_g) = \mathbb{E}_{x,x' \sim P_r}[k(x, x')] - 2\,\mathbb{E}_{x \sim P_r, y \sim P_g}[k(x, y)] + \mathbb{E}_{y,y' \sim P_g}[k(y, y')] \tag{2.19}$$

When applied to real data, the MMD distance is estimated using finite samples from the distributions. Given the samples $P_r \sim X = x_1, ..., x_n$ and $P_g \sim Y = y_1, ..., y_n$ the MMD estimator is estimated as follows:

$$\hat{M}_k(X, Y) = \frac{1}{\binom{n}{2}} \sum_{i \neq i'} k(x_i, x_{i'}) - \frac{2}{\binom{n}{2}} \sum_{i \neq j} k(x_i, y_j) + \frac{1}{\binom{n}{2}} \sum_{j \neq j'} k(y_j, y_{j'}) \tag{2.20}$$

Due to the sampling variance, even if the distributions are equal to each other ($P_r = P_g$), then $\hat{M}_k(X, Y)$ may not be zero. Thus, generally speaking, the lower the value of $\hat{M}_k(X, Y)$ the better and the closer the two distributions are to each other. Kernel MMD can be applied both in the pixel space and the feature space of a classifier network.

Kernel MMD works performs well in discriminability [53]. Additionally, its sample and computational complexities are low.

2.3.4 Frechet Inception Distance

The Fréchet Inception Distance (FID) metric was introduced by Heusel et al. [20]. FID embeds a set of generated samples into a feature space given by a specific layer of a classifier network. The Inception network is most commonly used for the classifier network, more specifically the Inception-v3 pooling-3 layer of the network. The resulting data are viewed as continuous multivariate Gaussian, where the mean and covariance are estimated for both the generated data and the real data. The Fréchet distance between these two Gaussians, also known as the Wasserstein-2 distance, is then used to determine the similarity between the two distributions.

The Wasserstein-2 distance is computed from the means and the covariances of the activation function values. It is calculated as follows, with subscript g denoting generated data and subscript r denoting real world data:

$$\text{FID} = ||\mu_g - \mu_r||_2^2 + \text{trace}(\Sigma_g + \Sigma_r - 2(\Sigma_g\Sigma_r)^{\frac{1}{2}}) \tag{2.21}$$

Despite only taking into consideration the first two moments of the distributions, the Fréchet inception distance metric performs well for GAN evaluation as shown in tests [53].

2.4 Fraud

Fraud detection can be treated as a binary classification problem or outlier detection problem since observations belong to one of two possible classes: positive or negative. The two classes can also be called true and false.

New observations are assigned to one of the two classes following some sort of decision rule. This decision rule can either be a model, or it can be some threshold value. Suppose one has a neural network model that provides as output 0 or 1, which is the class to which an observation is assigned. Here, the decision rule can be said to be the whole model. Whereas, when one has a model whose output is the probability of an observation belonging to one of the classes. Then, the decision rule is the probability value at which the observation is assigned to this class. This probability value, or threshold value, can be anywhere between 0 and 1. There are methods to determine which of the possible threshold values is optimal.

The classification results can be presented in the form of a confusion matrix, an illustrative example is presented in Table 2.1. It helps understand the ideas of true positive, true negative, false positive, and false negative observations.

		Predicted	
		Positive	Negative
Actual	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Table 2.1: Confusion table illustrative example.

The values from a confusion matrix are used for certain classification performance metrics. There are several different metrics that focus on different aspects of the classification performance of a decision rule. Depending on the use case of the classification

model the relevance of the different metrics changes. The most commonly used metrics are sensitivity, specificity, precision, and recall.

Sensitivity provides the proportion of the correctly predicted positive observations to the total number of actual positive observations, thus the sum of correctly predicted positive observations and incorrectly predicted negatives. The resulting value provides an indication of how many of the positive observations are actually predicted to be positive. The ideal value would be 1, as this means that no false negatives are predicted. Meaning that all the positive observations are predicted to be positive. It is also important for fraud detection, as an ideal value would mean that no fraudulent observations are missed and misclassified. Sensitivity is often called recall or true positive rate, and these names are often used interchangeably.

$$\text{Sensitivity/Recall} = \frac{TP}{(TP + FN)}$$

Specificity provides the proportion of correctly classified negative observations to the total number of actual negative observations, the sum of true negative and false positive observations. The resulting value provides an indication of how many of the predicted negative observations are actual negative observations. Frequently, instead of specificity, the false positive rate is used, this is equal to $(1 - Specificity)$. The ideal value for specificity is 1, as it means that all the observations that are predicted to be negative are indeed negative observations.

$$\text{Specificity} = \frac{TN}{(TN + FP)}$$

Precision provides the proportion of correctly classified positive observations to the total number of predicted positive observations. The resulting value provides an indication of how many of the observations that are predicted to be positive are relevant, i.e. are truly positive. The ideal value is 1, as it means that all the observations that are predicted to be positive are relevant, thus that no negative observations are predicted to be positive.

$$\text{Precision} = \frac{TP}{(TP + FP)}$$

If the classification decision rule is based on a threshold value, these performance metrics are used to find the optimal threshold value. The performance metrics are be calculated for each of the possible threshold values, and the results are displayed on plots. Depending on the application, different metrics are more important than others. For example, in medical applications not only is the true positive rate important but also the false negative rate. If a model is used for cancer detection, one want to have as few false negatives as possible, even at the cost of false positives. As this means that the fewer cancer cases will be missed. Thus in such an application sensitivity will be important.

One of these types of plots is the Receiver Operating Characteristic (ROC) plot. It displays the values of the false positive rate (FPR), (1-Specificity), on the x-axis, against the sensitivity on the y-axis. The ideal value for sensitivity remains the same, 1, whereas the ideal value of the false positive rate is 0. This means that a threshold value that leads to a perfect classification would be located in the top left corner of the plot, at the $(0, 1)$ coordinates. Usually, no threshold value will lead to a perfect classification. Therefore, the FPR and sensitivity values for the different threshold values are plotted on the ROC

graph, using a step function for better clarity. The point, or points, that are the closest to the top left corner of the plot are those that perform the best. If there are multiple points, and thus threshold values, that offer similar performance, then one chooses the threshold value based on which of the two metrics is deemed to be more important. ROC plots are indifferent to the balance of the classes, meaning that the number of observation belonging to each class does not have to be the same.

Another commonly used plot is the Precision-Recall curve. In this plot, the recall or sensitivity is plotted on the x-axis, whereas the precision is plotted on the y-axis. Again, the different threshold values are plotted according to their metric values, and a step function is used to plot them. Since the ideal value for both metrics is 1, a perfect threshold value would generate a point at the $(1, 1)$ coordinates. Since that is unlikely, the optimal threshold value, or values, will be those closes to this optimal point. Similarly to the ROC plot, if there are several threshold values that offer similar performances then one can select the optimal value based on which of the two metrics is deemed to be more important.

In the same fashion that the importance of the performance metrics depends on the application of the model, so does the importance of the plots. Depending on the application, and the nature of the data one of the plots will be more suitable for the selection of the threshold value.

Chapter 3

Experiments and Methods

In this section we will present and explain the experiments that will be performed. The experiments are designed in such a way as to investigate different behaviours and provide insights into specific areas of interest. Before we delve into the experiments, we need to provide information about the environment in which these experiments were run. This involves the hardware used, and the software environment necessary to run the experiments. This is necessary in order to provide context for the computational intensity and difficulty of the experiments, and in order to facilitate the repeatability of the experiments.

To train the models and perform the experiments a local machine, a desktop computer, was used. The machine in question has the following hardware specifications:

- Central Processing Unit (CPU): AMD Ryzen 2600x
- Random Access Memory (RAM): 32GB (2 x 16GB at 3200MHz)
- Graphics Processing Unit (GPU): Nvidia GTX1080 with 8GB VRAM
- Storage Drive: Samsung 970 Evo 500GB NVMe SSD

For the software, the machine is running the latest version of Microsoft Windows 10 as the operating system. In order to enable GPU-accelerated computing, the Nvidia Compute Unified Device Architecture (CUDA) tool kit - Version 11 was installed, along with the Nvidia CUDA Deep Neural Network library (cuDNN) - Version 8. The models and the experiments were written in Python, more specifically Python version 3.7. Within the Python environment, several different packages were used. These packages are listed below.

- Tensorflow 2 [1]
- NumPy [19]
- Pandas [36]
- Matplotlib [23]
- Seaborn [52]
- Python Optimal Transport [14]

The Tensorflow 2 package was used for the neural network models, both to build and to train them. It also allows to parallelise numerical operations so that they can be performed on the GPU, this was used for some calculations to speed them up. NumPy is a package that allows for easy matrix operations. Matplotlib and Seaborn are visualisation packages, they were used to create the results plots. Pandas is a package that allows to create data frames and perform operations on these data frames, it was used to create data frames for the results. Pandas works well in combination with Matplotlib and Seaborn. The Python Optimal Transport package was necessary in order to implement the Wasserstein distance metric.

Additionally, in order to generate illustrations of the GAN model architectures, the Graphviz [15] and PyDot packages were used. However, they are not required in order to train the models, perform the experiments, or visualise the results of the experiments.

3.1 Basic Distribution Experiment

Before we begin to perform any complex experiments, it is worthwhile to investigate the behaviour of the distance measures, Wasserstein distance, Kernel MMD, and Frechet distance, with a simple, controlled dataset. Since the distance measures require multivariate data we can generate samples from several multivariate datasets and calculate the distance between pairs of the samples.

For the experiment we will use 4 different multivariate distributions, each containing three variables. The first two variables of each multivariate distribution will follow the same Gaussian distributions, whereas the third random variable will follow a different Gaussian distribution for each of the multivariate distributions. The selected distributions are listed below.

- $X^A = (X_1 \sim N(1,1), X_2 \sim N(-1,1), X_3 \sim N(1,2))$
- $X^B = (X_1 \sim N(1,1), X_2 \sim N(-1,1), X_3 \sim N(1,4))$
- $X^C = (X_1 \sim N(1,1), X_2 \sim N(-1,1), X_3 \sim N(4,2))$
- $X^D = (X_1 \sim N(1,1), X_2 \sim N(-1,1), X_3 \sim N(-2,4))$

In order to perform the experiment, a reference distribution is selected: X^A. The experiment consists of n runs, during which samples, of size n, are generated from the multivariate distributions. During each run, two samples from the reference distribution are generated, and a single sample from each of the other multivariate distributions is generated. One of the two samples from the reference distribution is selected as the reference sample and the distance between it and each of the other samples is calculated. This step is then repeated n times. Taking two samples from the reference distribution allows us to see what the distance is between two samples from the same distribution.

3.2 Feature Space Selection Experiment

The aim of this experiment is to explore the effect of the choice of the feature space in which the distance metrics are calculated in, by comparing the original data space to feature spaces generated by the discriminators of different GAN models.

3.2.1 Feature Spaces

Data observations are usually either vectors or matrices of specific dimensions. These vectors or matrices can be said to within a space, whose number of dimensions is related to the dimensions of the vector or matrix. An n-dimensional vector representing an observation exists within an n-dimensional space, and occupies a specific position within this space. All potential other observations also exist within this space, at different location from the current observation. This space could be called a feature space. Each dimension within the space corresponds to a feature of the data, or observation. For many datasets, a feature can be a variable corresponding to a specific characteristic or measurement.

For many types of data, the individual variables in the data do not mean much. This is the case for images: an individual pixel value does not provide much information. Therefore, in such situations, the data is often transformed so that the variables in the transformed data have some meaning. This new space is then called the feature space, as each dimension refers to a feature in the data. These meaningful feature can now be used in models.

3.2.1.1 Dimensionality Reduction

Feature spaces generated by neural networks often have a large number of dimension, this is due to the fact that they often are the activation values taken from a specific layer in the neural network. It is often beneficial to perform dimensionality reduction on this data, in order to facilitate any further calculations that are to be performed. There are many different dimensionality reduction techniques. The two dimensionality reduction techiques that we are going to use were shown to be effective by Schreurs et al. [47].

Principal Component Analysis (PCA) [22] is a very effective dimensionality reduction technique. PCA aims to retain as much information contained within the data when going from the original space to the reduced space. This is achieved by taking linear combinations of the original variables in such a way as to maximise the amount of variation contained in the data. Each dimension in the reduced space is called a principal component, each subsequent principal component is perpendicular to the previous components whilst also maximising the variation that it captures.

Uniform Manifold Approximation and Projection (UMAP) [35] is a dimensionality reduction technique that has gained popularity in recent times. Part of this popularity is due to the possibility of creating nice and easy to understand visualisations. One of the main advantages of UMAP is that it can be used for general non-linear dimensionality reduction as it has no computational restrictions on embedding dimensions. It is constructed from a theoretical framework based in Riemann geometry and algebraic topology, including fuzzy topology.

3.2.2 Dataset

For experiment we will be using the MNIST dataset [10] [30]. This is an image dataset containing images of handwritten digits. The images are black and white, and 28-by-28 pixels in size. The fact that they are black and white means that there is only one depth dimension. Meaning that each individual observation has the following shape: (28, 28, 1). The digits range from 0 to 9, with the frequency of the digits being evenly distributed.

The MNIST dataset consists of two subset, the training set which consists of 50,000 observations and the training set which consists of 10,000 observations.

3.2.3 Experiment Design

In this experiment, we want to investigate the behaviours of the distance measures in different feature spaces. In order to perform an interesting comparison, we will expand the dataset. We will include two additional sets of observations. The first additional set is based on the MNIST test dataset. It consists of the same observations as the MNIST dataset, however the pixel values of the observations are shuffled. This means that the locations of the pixels in the images are randomly rearranged. The second additional dataset consists of images where the pixel values are randomly generated from the uniform distribution. Example observations from each of the three datasets are shown in Figure 3.1.

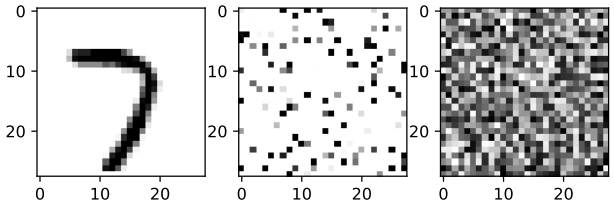


Figure 3.1: MNIST image, shuffled MNIST image, random data image.

The reasoning behind the selection of these datasets is somewhat simple. As humans, we can see that there is a significant different between the original images and the shuffled images. However, when looking at the distribution of the pixel values, this difference is not visible any more. Thus there is information in the positioning of the pixel values, and their positions relative to each other. To us, the randomly generated image is different than the other two. Despite also having random locations of the pixel value, it seems to have little in common with the shuffled image due to the difference in the distribution of pixel values.

The feature spaces that we will be comparing are the original pixel space, and the feature spaces created by the activation values of the penultimate layers of different GAN discriminator networks. The GAN discriminator networks should in theory learn to identify features that distinguish real images from fake images. The hope is that these learned features will cause different activation patterns in the real dataset, and the shuffled dataset. Since GANs are black box models, this is not guaranteed and therefore worthwhile investigating. We cannot know what exactly the GAN discriminator looks for in the images in order to determine whether an image is real or fake. Due to the similarity in the distribution of the pixel values, we can investigate the sensitivity of the distance measures to the arrangement of the pixels values.

Since we will be looking at samples of observations, we need to convert the sets of images into the shape of a 2-D matrix. This is done by flattening each image to a single vector composed of all the pixel values, and arranging these vectors into a matrix by rows. This results in a 2-D matrix where each row is an observation, and each column corresponds to an individual pixel, a variable. The discriminator network activation values are vectors, therefore the individual activation vectors only need to be arranged by rows

to create the corresponding sample 2-D matrix. These 2-D matrices are then used as inputs for the distance metric calculation functions.

Additionally, we will also investigate the two previously mentioned dimensionality reduction techniques, PCA and UMAP. This is to investigate how much information is lost when the feature space is reduced. The motivation behind this is the computational intensity of calculating the distance metrics. The feature spaces generated by neural network layers can be highly dimensional. This high dimensionality can have a negative impact on the speed at which the distances are calculated.

The experiment itself follows a simple structure which is described below.

1. Generate the samples between which the distances will be calculated:
   - Take sample $(x_1, ..., x_n)$ from the MNIST training set.
   - Take sample $(y_1, ..., y_n)$ from the MNIST training set.
   - Shuffle sample $(y_1, ..., y_n)$ to create sample $(s_1, ..., s_n)$ of shuffled observations.
   - Randomly generate sample $(r_1, ..., r_n)$ using $U(0, 1)$.
2. Transform samples into 2-D matrices.
3. If performing dimensionality reduction:
   (a) Train dimensionality reduction model on $(x_1, ..., x_n)$.
   (b) Transform all samples using the dimensionality reduction model.
4. Calculate distances between samples:
   - Distance($(x_1, ..., x_n)$, $(y_1, ..., y_n)$)
   - Distance($(x_1, ..., x_n)$, $(s_1, ..., s_n)$)
   - Distance($(x_1, ..., x_n)$, $(r_1, ..., r_n)$)
5. Save recorded distances.
6. Repeat steps 1 to 5, until no unused observations remain.

3.2.4 GAN models

Different GAN architectures and training methods may have different activation layer values. Therefore to investigate how important or how much of an effect these factors may have on the distance calculated within the activation layer feature space, we will be using three different GAN models, each following a different architecture, and different training method.

The three models that we are going to create, train, and then use the discriminators of are: DCGAN, LSGAN, and WGAN models. The specific details concerning their differences were covered in the literature review. The models were all trained using batch training, using batch sizes of 128 observations.

3.2.4.1 DCGAN

The DCGAN model that we are going to used in based on the model presented by Radford et al. [42].

The generator consists of series upsampling, convolutional, batch normalisation and ReLU activation layers that are repeated twice. The discriminator consists of convolutional, LeakyReLU, dropout, batch normalisation layers. The detailed architectures of the two networks are presented in the Appendix on Figure 6.1.

3.2.4.2 LSGAN

The LSGAN model that we are going to be using is based on the model presented by Isola et al. [25].

The generator mode consists of LeakyReLU, batch normalisation, and dense layers. The discriminator consists of dense and LeakyReLU layers. The detailed architectures of the two networks are presented in the Appendix on Figure 6.2.

3.2.4.3 WGAN

The WGAN model that we are going to be using is based on the model presented by Arjovsky et al. [3] .

The generator model consists of unsampling, convolutional, batch normalisation, and ReLU activation layers. The discriminator model consists of convolutional, LeakyReLU, and dropout layers. The detailed architectures of the two networks are presented in the Appendix on Figure 6.3.

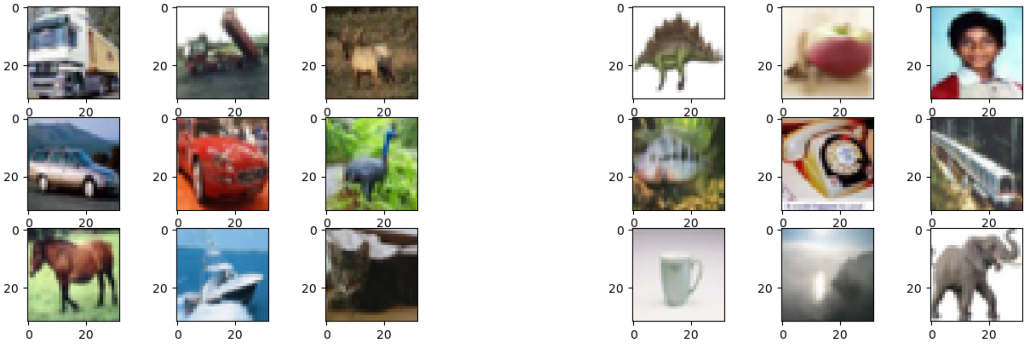
3.3 Fraud Detection Experiments

In the experiments in this section we want to investigate the possible applications of the distance metrics within fraud detection scenarios.

3.3.1 Data

For the experiments in this section we will be using the CIFAR10 [29] and CIFAR100 [29] datasets. These datasets both contain images of the same size. They are coloured images, 32-by-32 pixels in size. Since they are coloured, they have three depth dimensions corresponding to the RGB values for each pixel. This means that each individual observation is of the size (32, 32, 3). What differentiates the two datasets are the categories of subjects shown on the images. The subjects of the images in the CIFAR10 belong to 10 categories. The subjects of the images in the CIFAR100 dataset belong to 100 categories. What is particularly important for our experiments, is that there is no overlap between the subject categories of the two datasets. Some categories are similar in the way a bus and tractor are considered to be similar, since they both are vehicles, but they remain two clearly different types of vehicles. Example images from both datasets are shown in Figure 3.2.

For our experiments we will be using the CIFAR10 dataset as the *true* dataset, containing the 'real', or 'non-fraudulent', observations. Whereas the CIFAR100 dataset will



(a) CIFAR 10 example images.

(b) CIFAR 100 example images.

Figure 3.2: Dataset demonstration.

be the $false$ dataset, containing the 'fake', or 'fraudulent', observations. For our experiment we need the $true$ data to be split into $true_{known}$ and $true_{unknown}$ datasets. The $true_{known}$ will be used to train the GAN models, whose discriminators will be used to generate feature spaces. This dataset will also be used to sample reference observations, and to generate reference distributions. The $true_{unknown}$ dataset will be used to obtain new $true$ observations for the experiments. The $false$ dataset will be called the $false_{unknown}$ dataset, as it is not used to generate the feature spaces. Since we work under the constrain that no fraudualent observations are available for training.

3.3.2 Feature Space Models

For the experiments that we will perform with this dataset, we will be calculating the distances within features spaces. Thus, we need to select or create these feature spaces.

3.3.2.1 GAN Model

We will use a DCGAN model to generate a GAN discriminator feature space. The results from the feature space results, in the results section, showed little difference between the different architectures. The model with the best quality generate images was the DCGAN model. The architecture of the DCGAN model used for this experiment is based on the architecture used in the feature space experiment, modified to work with the different size of the images. The detailed architecture can be seen in the appendix in Figure 6.10.

3.3.2.2 Inception Model

The InceptionV3 [49] model is a very well known classification model. We already mentioned this model when discussing the Inception Score in the GAN metrics in the literature review. It is a classification model with pre-trained weights using the ImageNet [11] dataset.

3.3.3 Experiment Designs

There are two experiments that we will perform with these dataset. Both use the previously mentioned CIFAR datasets and the feature spaces of the InceptionV3 and DCGAN discriminator models. One investigates multiple observation fraud detection, whereas the other investigates single observation fraud detection.

3.3.3.1 Multiple Observation Experiment Design

In this experiment we focus on large samples of observations, and investigate how the distance measures behave within this type of fraud detection setting. In order to investigate how sensitive the distance measures are to the existence of fraudulent observations within the samples, we will calculate the distance between a reference sample and samples of new observations. These samples of new observations will include different proportions of non-fraudulent and fraudulent observations. We will also investigate how the size of the sample affects the behaviour of the distance values.

Below is a step by step description of the experiment:

1. Reference sample $(X_1, ..., X_n)$ is taken from the $known_{true}$ population.
2. For the given proportion p_i of fraudulent observations the following new samples are generated:
   - $n * p = m$ fraudulent observations $(Y_1, ..., Y_m)$ from the $unknown_{false}$ population.
   - $n*(1-p) = n-m$ non-fraudulent observations $(Y_{m+1}, ..., Y_n)$ from the $unknown_{true}$ population.
3. Combine the two new samples to form one new sample $(Y_1, ..., Y_n)$.
4. If dimensionality reduction is being applied:
   (a) Generate dimensionality reduction model, reducing the feature space from n to r dimensions ($r << n$).
   (b) Train dimensionality reduction model on reference sample $(X_1, ..., X_n)$.
   (c) Transform the reference sample and new sample with the dimensionality reduction model.
   (d) Complete following steps using reduced samples $(X_1^r, ..., X_n^r)$ and $(Y_1^r, ..., Y_n^r)$.
5. Calculate distance between the reference and new samples for each distance measure.
6. Repeat steps 2 to 5 to build distributions of distance values, and for each level of $p_i = (p_1, ..., p_k)$., and sample sizes.

The results from this experiment will help create a better understanding about the type of environment in which a fraud detection model based on the distance measures could be used for. We will be able to understand what type of relationship exists between the distance values and the proportion of fraudulent observations in the samples. This could be linear, exponential, or some other type. The type of relationship can have an effect on the possible applications and general viability of the model idea.

3.3.3.2 Single Observation Experiment Design

In this experiment we investigate the possibility of performing fraud detection on single observations using the distance measures, in the DCGAN discriminator feature space.

The distance metrics that we have been using and are going to use for this experiment are made with multivariate data samples in mind. This means that they require the sample between which the distances are calculated to have $n > 1$ observations and $m > 1$ variables. Therefore, in order to perform fraud detection on single observation we will have to turn the single observation feature vector into a multivariate sample that satisfied these conditions. There are several different ways in which this transformation could be performed. The feature vector can be slip into several equal length vectors and those can be treated as individual observation, together forming a sample from a multivariate distribution. Another idea would be to repeat the single observation several times, and add some noise so that each observation would become slightly different. This first method is much easier than the first one, and it does mean that the information in the data is not modified. The second idea seems maybe more sensible, but it would be very complicated to implement. Each variable is likely to have a different variance. Thus applying the same type of noise to every variable value could result in the removal of the information contained in the observation. One would first have to look at sample of different observations in order to determine the variance of the value of each variable. Then, these variances would have to be used to generate a map for the noise that should be applied to the repeated single observation. Therefore, we will try this experiment using the method where the observation vector is split into several parts, and the parts are rearranged to create a matrix.

For this experiment we use three sets of observations. The first set is the reference set, the second set is the known set, and the third set is the set of unknown, or new observations. The reference and known sets are used to generate a reference distribution of average distance values. This is by calculating the average of the distances between an observation from the known set, and every observation in the reference set. The unknown and reference sets are used to calculate the distances for the new/unknown observations. Here the average of the distances between an observation from the new set and each of the reference set observations is calculated. These new distances will then be compared to the reference distance distribution during the classification process. The experiment is described step by step below.

1. Take a random set of n reference observations $(R_1, .., R_n)$ from the $known_{true}$ population.

2. Take a random set of q known observations $(X_1, ..., X_q)$ from the $known_{true}$ population, that does not overlap with the set of reference observations.

3. Take a random set of m new observations $(Y_1, ..., Y_m)$ from the $unknown_{true}$ and $unknown_{false}$ populations.

4. Transform single observations, activation value vectors, into 2-D matrices:

   - Take X_i (or Y_i or R_i) which is vector of length m.
   - Transform X_i (or Y_i or R_i) into matrix of size $(k, m/k)$.

5. If dimensionality reduction is applied:
   (a) Generate a dimensionality reduction model for each R_i matrix.
   (b) Train the dimensionality reduction models on their respective R_i matrices.
6. To generate the reference distance distribution, for each Z_i in the set of known observations:
   (a) Calculate the distance between Z_i and every R_i. (Or their reduced matrices, using the R_i trained reduction model).
   (b) Store the average of these distance.
7. The stored average distances form the reference average distance distribution.
8. Calculate the average distance for each Y_i in the unknown observations.
   (a) Calculate the distance between Y_i and every R_i. (Or their reduced matrices, using the R_i trained reduction model).
   (b) Store the average of the distances.
9. The stored average distances are the unknown average distances.
10. Classify each Y_i from the unknown observations:
    (a) Take the average distance of Y_i.
    (b) Take the average distance value from the reference distance distribution that corresponds to the threshold T_i.
    (c) If the average distance of Y_i is greater than T_i, then Y_i is classified as fraudulent. Else, it is classified as non-fraudulent.

The step by step description is a somewhat simplified version of the real experiment for better ease of understanding. In practice, the experiment is performed simultaneously for all three distance measures. The classification process is separate from the distance calculations. And, the classifications are performed using a range of threshold values.

Chapter 4

Results

4.1 Basic Distribution

In this experiment we calculated the distances between pairs of samples from 4 different multivariate distributions: X^A, X^B, X^C, X^D. After running the experiment, it makes sense to show the distributions of the values generated for the X_3 variables in the four samples. These distributions can be seen in Figure 4.1.

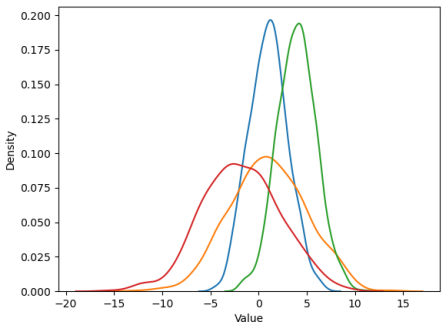


Figure 4.1: The different distributions of the X_3 random variables.

The results from the experiment can be seen in Table 4.1. It includes the average distance between the distributions for the 100 samples, and the 95% confidence intervals of these distances. We can see that most of the results follow the expected behaviour. The mean distance between the least different distributions is the smallest, whereas the mean distance between the most different distributions is the largest. However, this is not the case for every metric. For the Kernel MMD metric, the average distance between the $N(1, 2)$ and $N(4, 2)$ distributions is greater than than the average distance between the $N(1, 2)$ and $N(-2, 4)$ distributions. For the remaining distance metrics, the distances increase at each step. For all the distance metrics, the average distances increase more in the the $N(1, 2)$ and $N(4, 2)$, than the $N(1, 2)$ and $N(1, 4)$ comparison compared to the distance between two $N(1, 2)$ samples. This suggests that a change of 2 in the mean makes a distribution more different than increasing the standard deviation by 2. This makes sense, as during the increase of the standard deviation most of the distribution remains in the range of the original distribution. Whereas, when the mean is changed by 2, more of the distribution moves out of the range. When looking at the 95% confidence

intervals, we see that for the Wasserstein and Kernel MMD distances, the first three distributions do not overlap. Whereas, for the Frechet distance only the first two do not overlap. This may suggest that the Frechet distance varies more than the other two. For all the distance measures, the last two distributions overlap. This may suggest that the previously mentioned Kernel MMD behaviour is not that unusual. It is important to remember that the samples used were from multivariate distributions, where the values of first three random variables (X_1, X_2, X_3) were also randomly generated for each sample. The values generated for these variables will also have an effect on the distance results.

Distance	Wasserstein	kMMD	Frechet
$N(1,2)$ to $N(1,2)$	0.89 (0.791, 1.034)	0.077 (0.041, 0.132)	0.229 (0.058,0.572)
$N(1,2)$ to $N(1,4)$	1.839 (1.403, 2.345)	0.254 (0.173, 0.341)	4.167 (2.168, 7.082)
$N(1,2)$ to $N(4,2)$	3.045 (2.591, 3.651)	0.517 (0.438, 0.618)	9.152 (6.51, 13.168)
$N(1,2)$ to $N(-2,4)$	3.320 (2.63, 4.17)	0.455 (0.344, 0.560)	13.639 (8.199, 21.965)

Table 4.1: Multivariate distribution distance metric performance. Wasserstein and Frechet increase for each change in distribution. Kernel MMD decreases when both mean and variance are changed, against just mean.

4.2 Feature Space Selection Experiment

Now we turn to the results of the feature space experiment. In these results we can compare the distribution of the known-to-unknown sample distances to the distributions of the known-to-random and known-to-shuffled sample distributions, within the feature spaces of different GAN discriminator networks.

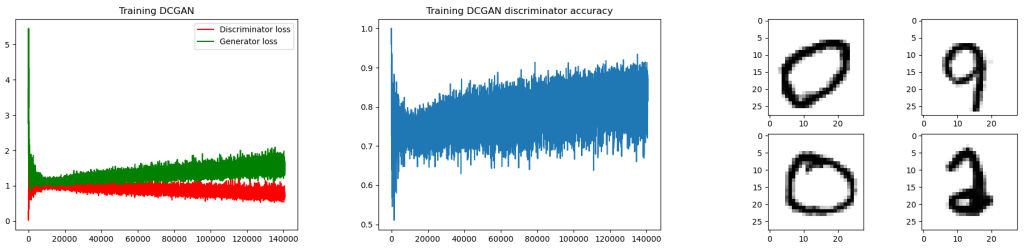
4.2.1 GAN Model Training

We use three different GAN models in order to obtain three different discriminator feature spaces. Before we are able to use the discriminators for the feature spaces, we need to train these GAN models. For our application, it is not necessary for the GAN models to be perfect, they do not need to generate perfect new observations. Since we are going to be using the discriminator network for the feature space, the performance of the generator network is not of great importance. Of course, the discriminator needs to be trained enough to identify features in the data. Thus we are not going to train the GAN models until they are 'perfect'. Once a model is able to generate observations that somewhat resemble the target dataset, the discriminator should hopefully be trained well enough to have learned features, and to generate an adequate feature space. Since obtaining 'perfect' GAN models can be very difficult, these relaxed requirements mean that an appropriate network should not be too difficult to create and train.

All models were trained for the same number of epochs, thus they were trained using the same number of training steps. During the training process of each model, the MNIST training dataset, consisting of 60000 images, was fed through the networks 300 times.

4.2.1.1 DCGAN

In Figure 4.2 we see three different plots providing information about the training process of the DCGAN model. We can see the loss value plots for both the generator and the discriminator networks. The losses fall drastically at the very start of training and later they start slowly increasing. We can also see the evolution of the discriminator accuracy during the training process. At first it varies a lot, then converges and slowly starts trending up. Finally, we also see some sample observations generated by the generator network, which look reasonably good.

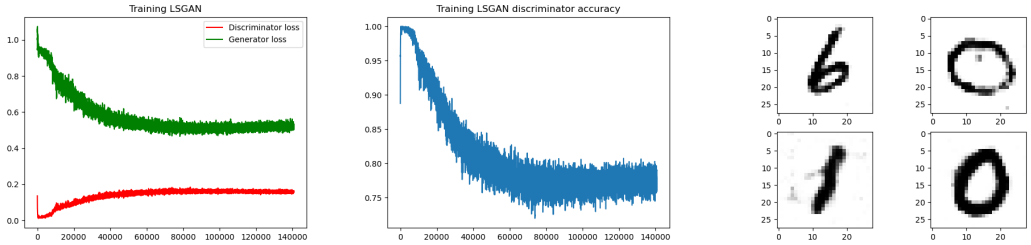


(a) Network losses (b) Discriminator Accuracy (c) Generated Examples

Figure 4.2: DCGAN training summary.

4.2.1.2 LSGAN

In Figure 4.3 we see the training summary for the LSGAN model. The losses follow smooth trends, and stay around the same value in the latter half of the training process. The discriminator accuracy has relatively low variance, and follows a downward trend, until is stabilises between 0.75 and 0.81. The generated examples look worse than the DCGAN examples.

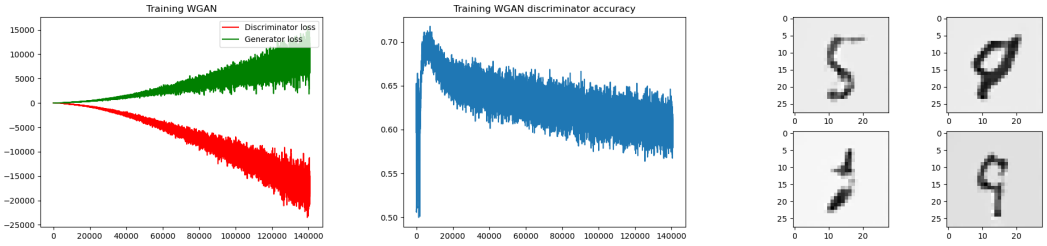


(a) Network losses (b) Discriminator Accuracy (c) Generated Examples

Figure 4.3: LSGAN training summary.

4.2.1.3 WGAN

In Figure 4.4 we see the training summary of the WGAN model. We can see that the losses are diverging. The discriminator accuracy varies a lot at the start and actually increases for a short amount of time. Afterwards, it stabilises and starts to follow a downward trend. The generated examples are different than those of the other two models as their backgrounds are grey rather than white, and additionally their quality is quite poor.

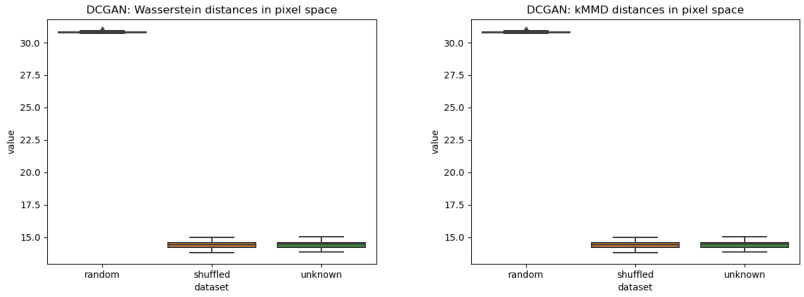


(a) Network losses. (b) Discriminator Accuracy. (c) Generated Examples

Figure 4.4: WGAN training summary.

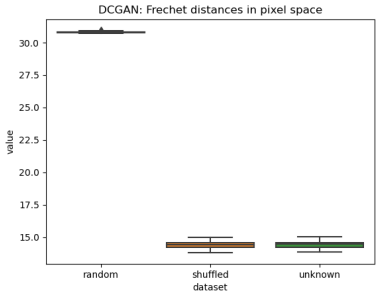
4.2.2 Distance Results

With the trained GAN models, it is possible to create the discriminator feature spaces by removing the last, dense, layer of the network. Then obtain the feature space representations of the observations and calculate the distances within these feature spaces. For the dimensionality reduction methods, both PCA and UMAP, 25 components were retained.



(a) Wasserstein

(b) Kernel MMD



(c) Frechet

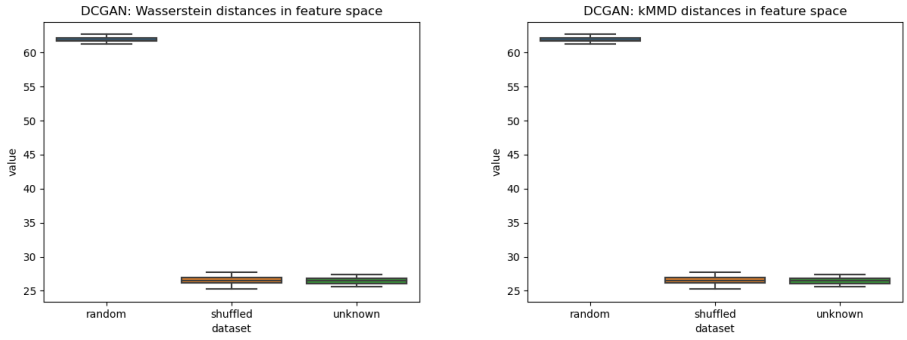
Figure 4.5: Pixel space results, distance for random set much higher than unknown and shuffled. Little difference between unknown and shuffled distributions.

4.2.2.1 Pixel Space

In Figure 4.5 we can see box plots of the distances between the known observations and observations from each of the unknown, shuffled, and random datasets. Meaning that we have distributions for the known-to-unknown, known-to-shuffled, and known-to-random distances. We can see that the distance to the random dataset are the greatest. The unknown and shuffled distance distributions are similar.

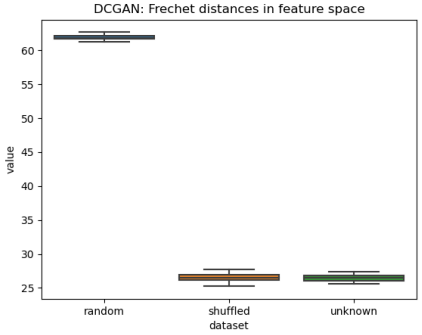
4.2.2.2 DCGAN Feature Space

In Figure 4.6 we can see the distance results in the DCGAN feature space. We can see that they are similar to the results within the pixel space. We would have hoped that the distance distributions of the unknown and shuffled datasets would differ more within the DCGAN feature space than within the pixel space. However, this is not the case. They remain very similar. The shuffled distance distributions seems to have slightly greater variance compared to the unknown distribution.



(a) Wasserstein

(b) Kernel MMD

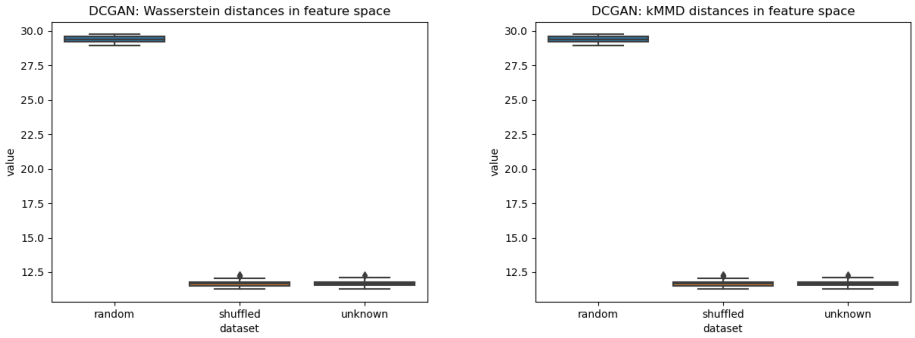


(c) Frechet

Figure 4.6: DCGAN feature space results, unknown and shuffled distance distributions remain very similar.

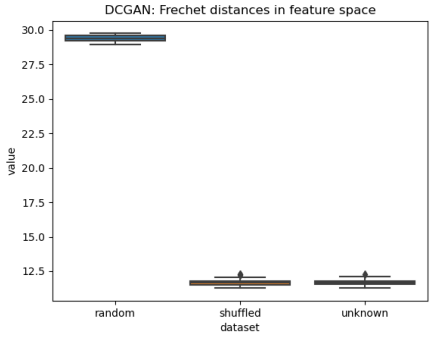
4.2.2.3 LSGAN Feature Space

Figure 4.7 shows the distance results when calculated in the LSGAN feature space. They are very similar to the DCGAN results. Again, the distance distribution of the random observations is much greater than the distance distributions of the other two. The unknown and shuffled distance distributions are very similar once again.



(a) Wasserstein

(b) Kernel MMD



(c) Frechet

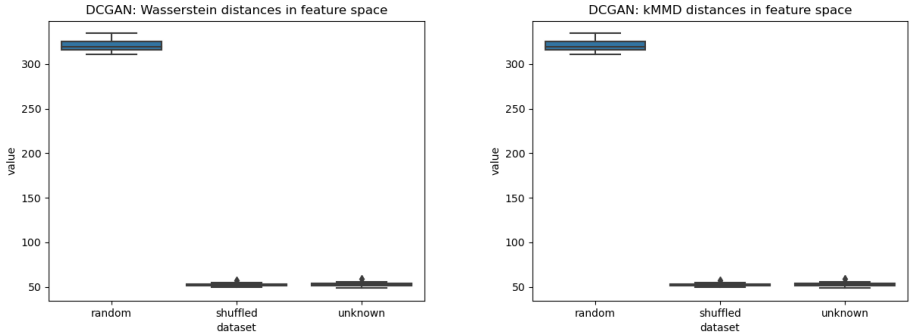
Figure 4.7: LSGAN feature space results, similar to DCGAN results. Little difference between unknown and shuffled distances.

4.2.2.4 WGAN Feature Space

Figure 4.8 shows the distance results for the WGAN feature space. The results are very similar to the DCGAN and LSGAN results in the behaviour of the different distance distributions.

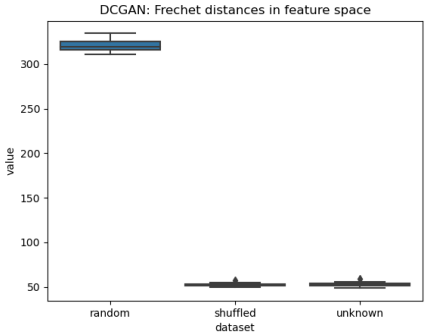
4.2.2.5 Feature Space Reduction

The results for the experiment when dimensionality reduction techniques were used are very similar to the behaviours in the full feature spaces. No observations of interest can be made from those results. The plots for the results in the PCA reduced space are shown in the Appendix in Figures 6.4, 6.5, and 6.6. The results for in the UMAP reduced space are also shown in the appendix, in Figures 6.7, 6.8, and 6.9.



(a) Wasserstein

(b) Kernel MMD



(c) Frechet

Figure 4.8: WGAN feature space results, similar to DCGAN and LSGAN results. Little difference between unknown and shuffled distances.

4.2.3 Results Summary

Overall, the results of this experiment are not encouraging. We were hoping that in the feature space of a GAN discriminator, the known-to-shuffled distances would become greater than the unknown-to-known distances, when compared to the results in the pixel space. The results showed little difference in the behaviours of these distance distributions depending on the feature space. This does not mean that the idea of using a GAN discriminator feature space should be discarded. The distance to the random dataset remained high within the GAN feature spaces, thus that information was not lost. It is possible that the MNIST dataset and this experiment were not suited to investigate the problem well. The dataset is quite simple, being black and white, and low resolution. The shuffled images are shuffled randomly. So, it is possible that the images generated after shuffling could contain some of the features that the discriminator is trained to detect, as unlikely as that may be.

4.3 Fraud Detection Experiments

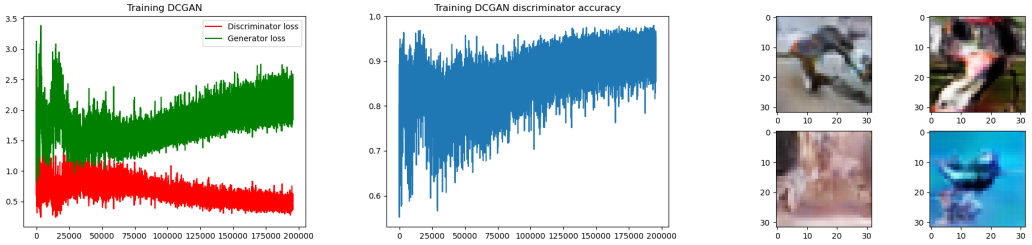
Here we will present the results from the fraud detection experiments. In these experiments we use a DCGAN discriminator features space, and the Inception network feature space. The datasets that we are using are the CIFAR10 and CIFAR100 datasets.

4.3.1 GAN Model Training

Similarly to the feature space experiment, we need to train the GAN model whose discriminator is used to generate the feature space. Similarly to the feature space experiment, our goal is not to create a 'perfect' GAN model. Considering that the CIFAR10 dataset is more complex than the MNIST dataset, creating a perfect GAN model would be even more difficult. The results from the previous experiment did not show any significant differences between the features spaces of different GAN models. For this experiment we will be using a DCGAN model.

4.3.1.1 DCGAN Training

The DCGAN model was trained using 500 passes. Meaning that the CIFAR10 training dataset of 50,000 images was fed through the network 500 times during the training process. We can see training summary plot in Figure 4.9. We can see that the loss values are changing, first both decreasing then diverging. The discriminator accuracy varies a lot and does not seem to be decreasing. However, the generated images look quite promising.



(a) Losses (b) Discriminator Accuracy (c) Generated Examples

Figure 4.9: DCGAN training summary.

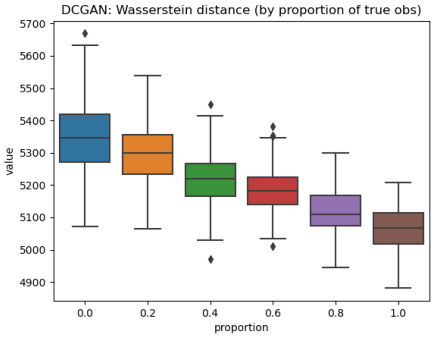
4.3.2 Multiple Observation Experiment

In order to make the results between the different feature spaces fair to compare, a random seed value of 5073 was used for each experiment. This means that the same observations were randomly selected each time, allowing for a fair comparison between the different feature spaces. Since the results for all the distance measures are calculated during the same experiment, they use the same observations.

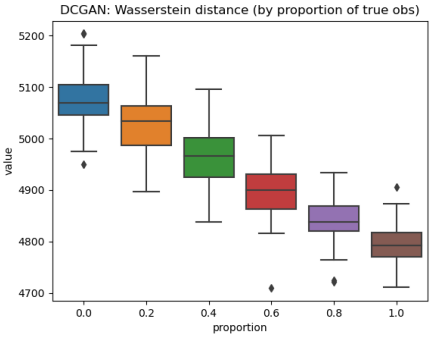
We perform the experiment with sample sizes of 100, 250, and 500 observations. For the dimensionality reduction techniques, 25 components were used for both PCA and UMAP. This allows us to see how the behaviours of the distance metrics change as the sample size changes. We present the results for the sample sizes 100 and 250 together in groups of different features spaces for the same distance measure. This allows us to easily

compare the effect of the different dimensionality reduction techniques, and the effect of the increased sample size. We present the results for sample size 500 in groups where the results for each distance metric within the same feature space are shown together. This allows us to have a better look at the differences in behaviour between the distance measures.

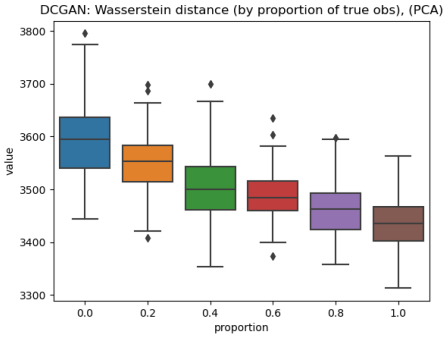
4.3.2.1 DCGAN Feature Space



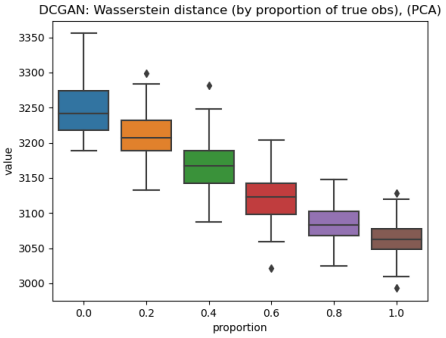
(a) Wasserstein (n=100)



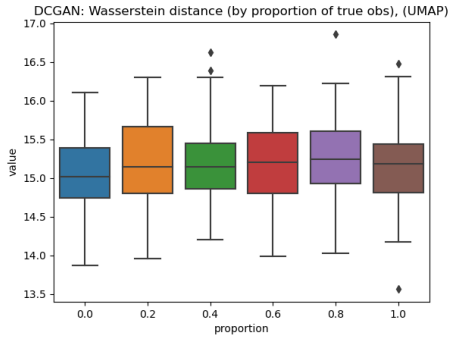
(b) Wassestein (n=250)



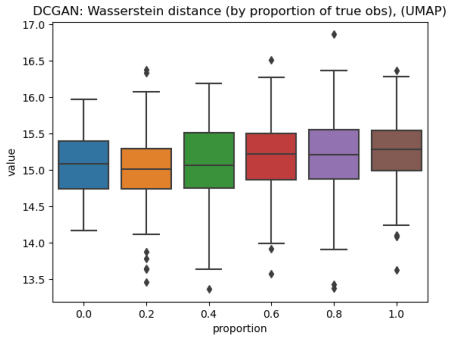
(c) Wasserstein (PCA, n=100)



(d) Wasserstein (PCA, n=250)



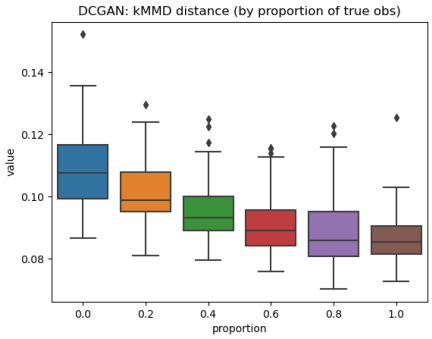
(e) Wasserstein (UMAP, n=100)



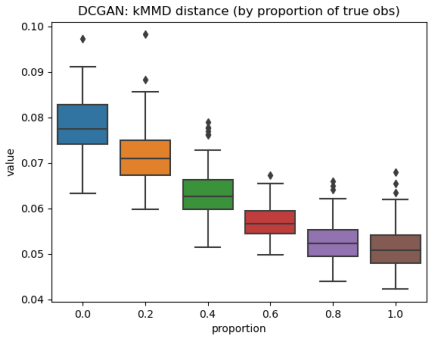
(f) Wasserstein (UMAP, n=250)

Figure 4.10: DCGAN, Wasserstein distance (n=100 & 250). Medians follow linear trend. Better separation at higher sample size. PCA reduction has little effect, UMAP reduction creates poor results.

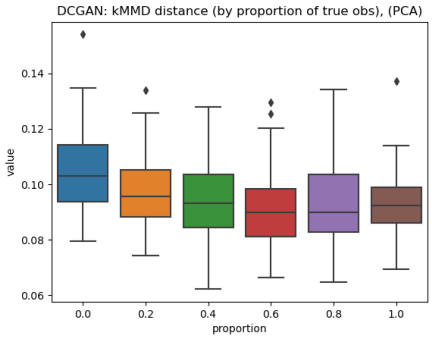
We can see the results for the Wasserstein distance measure in Figure 4.10. We see that the increase in sample size from 100 to 250 improves the separation between the distance distributions. The median distances seem to follow a linear pattern as the proportion of true observations in the sample falls. We can see that using PCA dimensionality reduction does not have much of an impact on the behaviours of the distance distributions. The UMAP reduction results are poor in comparison to the other two feature spaces. There is no clear then in the medians.



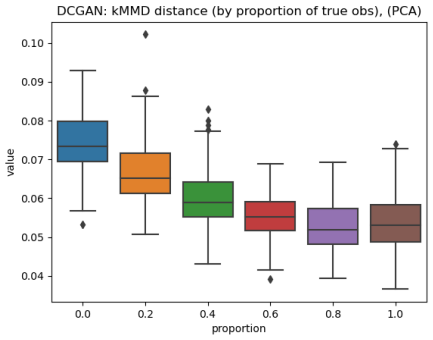
(a) Kernel MMD (n=100)



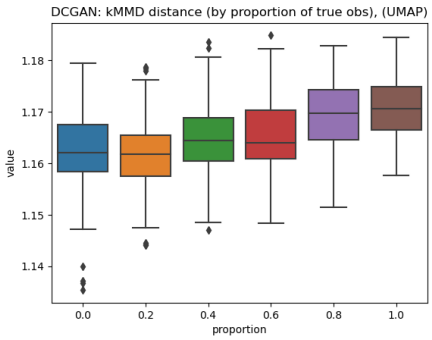
(b) Kernel MMD (n=250)



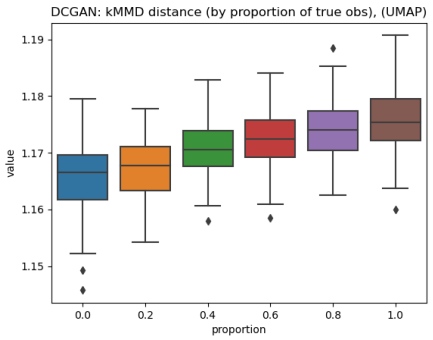
(c) Kernel MMD (PCA, n=100)



(d) Kernel MMD (PCA, n=250)



(e) Kernel MMD (UMAP, n=100)

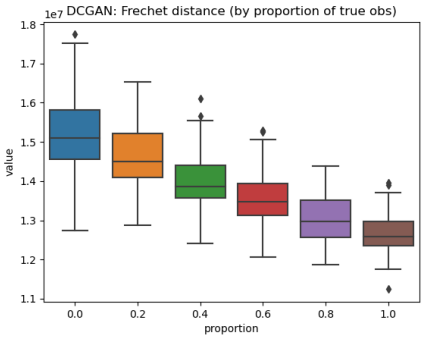


(f) Kernel MMD (UMAP,n=250)

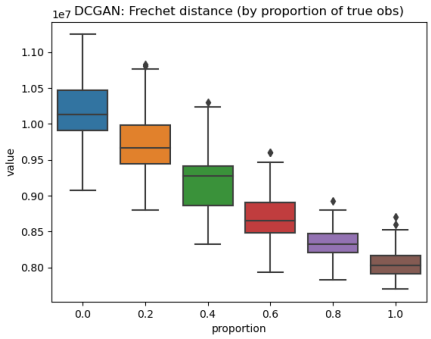
Figure 4.11: DCGAN, Kernel MMD distance (n=100 & 250). Medians follow exponential trend. PCA reduction has little effect. UMAP reduction relationship inverse of expectation: fake samples closer than real samples.

Figure 4.11 shows the results for the Kernel MMD distance measure. The distance distributions become more separated as the sample size increases from 100 to 250. The medians of distance distributions seem to follow an exponential trend as the proportion of true observation falls, this is more visible when the sample size is 250. This trend is different from the Wasserstein distance results where the relationship was more linear. The exponential trend also becomes more apparent in the PCA reduced space. The UMAP reduced space results are strange. The trend is the inverse of what one would expect. The distance decreases as the proportion of false observations in the sample increases. Since this behaviour only occurs in the UMAP reduced space, it most likely is related to this reduction. There may be some interaction between Kernel MMD and UMAP reduction which causes it.

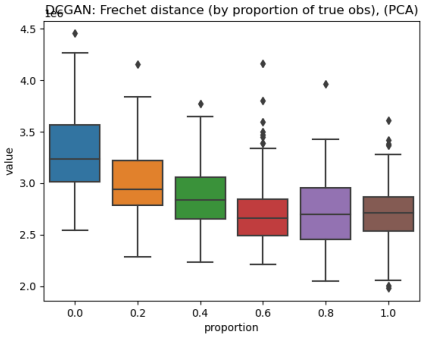
Figure 4.12 shows the results for the Frechet distance measure. Again, the distance distributions exhibit more separation as the sample size increases from 100 to 250. The behaviour of the medians is quite interesting. For the full feature space they seem to follow a linear trend, similar to what the Wasserstein results look like. Whereas, in the PCA reduced space the medians follow an exponential trend. The UMAP reduced space results are poor.



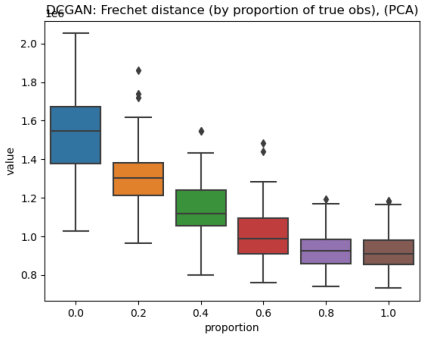
(a) Frechet (n=100)



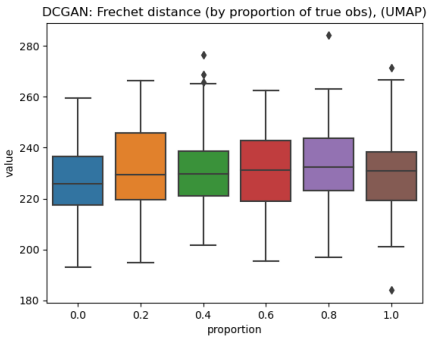
(b) Frechet (n=250)



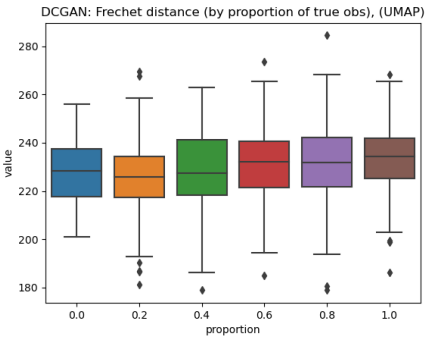
(c) Frechet (PCA, n=100)



(d) Frechet (PCA, n=250)

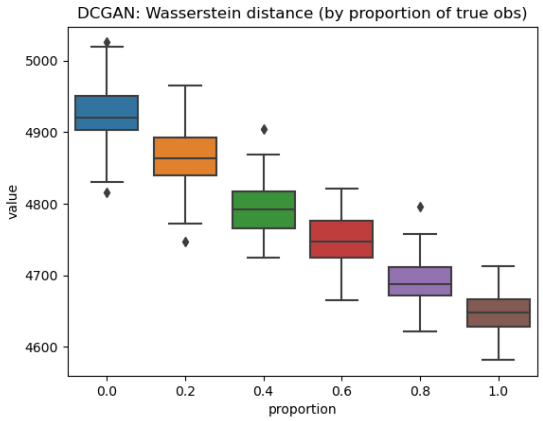


(e) Frechet (UMAP, n=100)

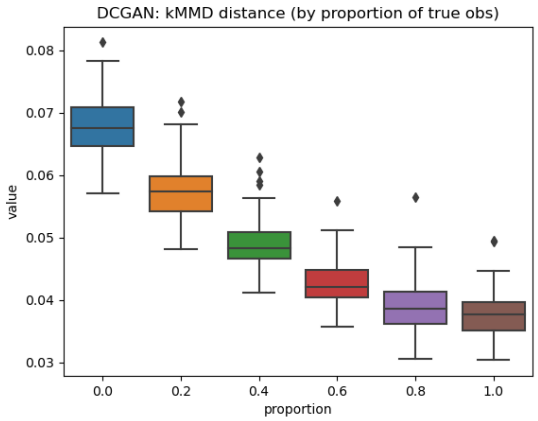


(f) Frechet (UMAP, n=250)

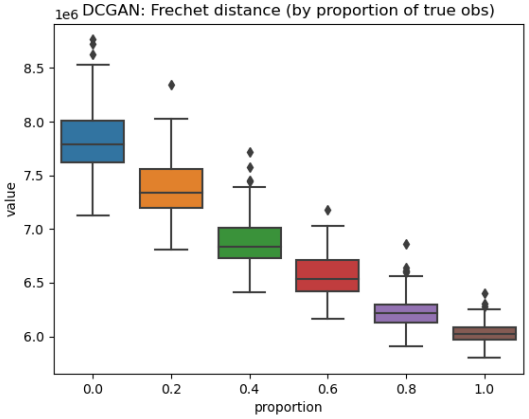
Figure 4.12: DCGAN, Frechet distance (n=100 & 250). Medians follow linear trend in full space, exponential trend in PCA reduced space. UMAP reduction creates poor results.



(a) Wasserstein



(b) Kernel MMD

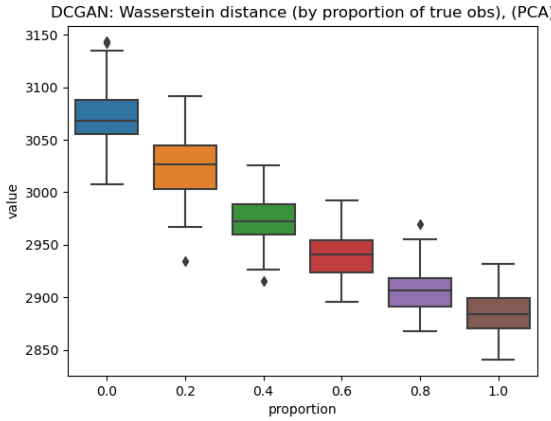


(c) Frechet

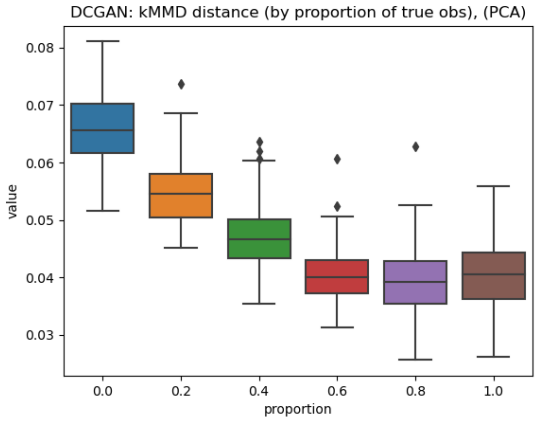
Figure 4.13: DCGAN all distances (n=500). All have clear trends and good separation. Wasserstein linear, Kernel MMD exponential, Frechet somewhat linear.

Figure 4.13 shows the results, in the full DCGAN feature space, for all three distance measures when the sample size is 500. The behaviours of the medians are similar to those when the sample size was 250: the Wasserstein trend is linear, Kernel MMD exponential, and Frechet somewhat in between. Looking at the boxes in the plots, we can use the box where the proportion of true observations is 1 as a baseline. Then, we can look at which proportion level the minimum of the box of said proportion level is greater than the maximum of the box when the proportion level is 1. We see that for the Wasserstein and Frechet distances this happens at the proportion of 0.4, whereas for the Kernel MMD distance, it happens at the proportion level of 0.2. The Wasserstein distance results also seem to have the fewest outliers, although some of those outliers are low values. Whereas, for the other distance measure the outliers are high values.

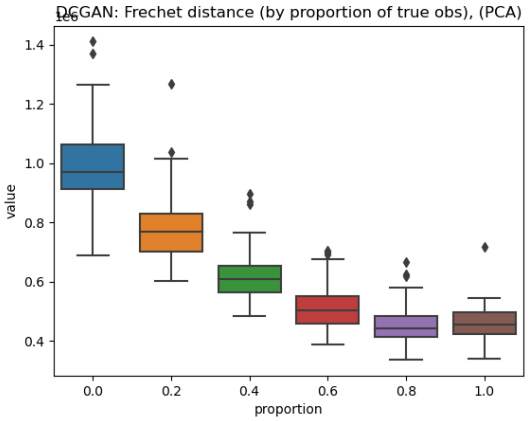
The results for the three distance measures, with sample size 500, but calculated in the PCA reduced space are shown in Figure 4.14. We can see that the Frechet distance medians follow a more exponential trend than in the full feature space. The Wasserstein distance trend remains linear. We can once again look at the minimum and maximums. For the Wasserstein and the Frechet distances, the minimum at level 0.2 is greater than the maximum at level 1. Whereas for the Kernel MMD distances, no minimums are greater than the maximum at level 1. Comparing this to the full feature space results, for each of the distance metrics the proportion of fake observations required for to pass the test has increased. This is likely due to the lower amount of information contained in the reduced data. It seems that the Kernel MMD distance is the most affected by the PCA reduction, changing by two proportion levels. Whereas the Wasserstein and Frechet metrics only change by one proportion level.



(a) Wasserstein



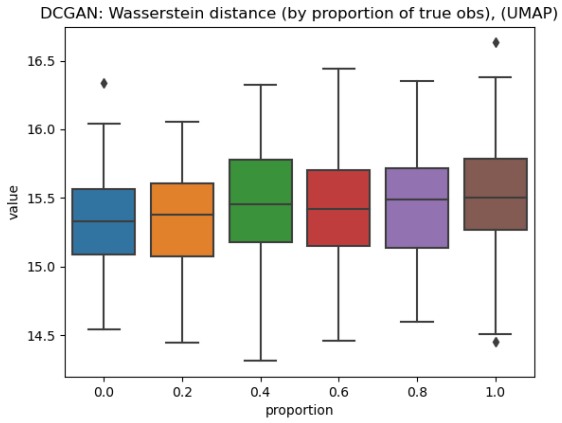
(b) Kernel MMD



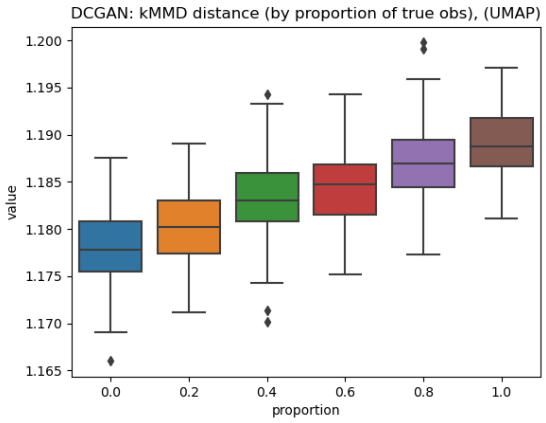
(c) Frechet

Figure 4.14: DCGAN, PCA reduction, all distances (n=500). Wasserstein trend remains linear. Kernel MMD and Frechet have exponential trends. For both Kernel MMD and Frechet the 0.8 and 1 proportion distributions are very similar to each other.

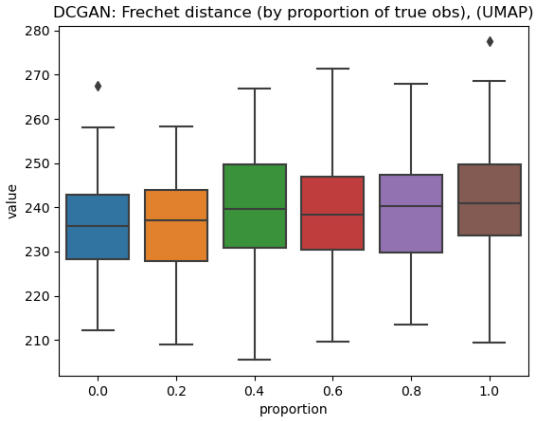
Finally, we can see the results for the UMAP reduced space at a sample size of 500 in Figure 4.15. The results are poor, with the Kernel MMD distance again exhibiting the inverse of the expected trend. The distance distributions for the other two distance measures do not follow such a strong trend. One could argue that the Wasserstein and Frechet distributions do faintly follow an inverse trend too. This is very interesting behaviour, but not helpful for the fraud detection purpose that we are investigating.



(a) Wasserstein



(b) Kernel MMD



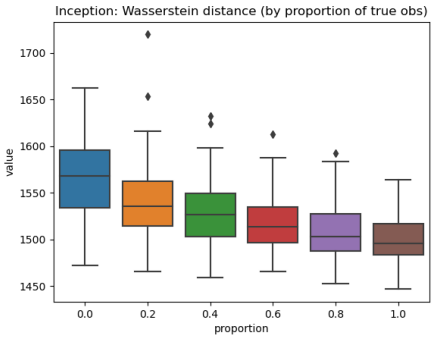
(c) Frechet

Figure 4.15: DCGAN, UMAP reduction, all distances (n=500). Wasserstein and Frechet relatively stable across proportion levels. Kernel MMD relationship is inverse to what is expected.

These results provide us with several interesting observations. First of all, we observe the importance of have large enough samples. At each increase in the sample size, the distance distributions became more separated. We also observe that the medians of distance distributions follow different trends for the different measures, and these trends also change when going from the full feature space to a reduced feature space. The Wasserstein metric seems to be least affected by a reduction of the feature space, whereas the Frechet metric seems to be the most affected. We also observe that the UMAP feature space reduction loses a lot of information, and causes the distributions to follow an inverse trend.

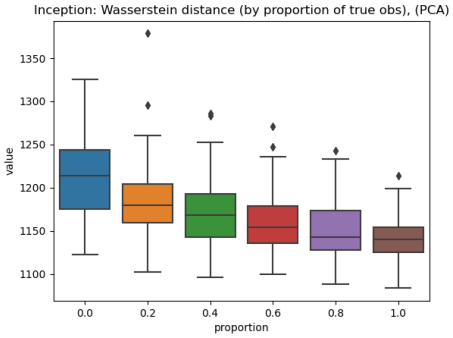
4.3.2.2 Inception Feature Space

We will now look at the results from the same experiment using the same random seed, except that the distances were calculated in the Inception network feature space. The presentation of the results follows the same sequence as the one used to present the DCGAN feature space results.

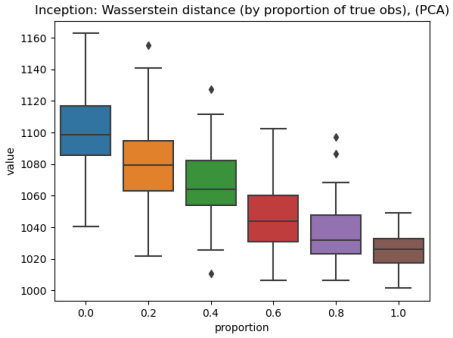


(a) Wasserstein, n=100

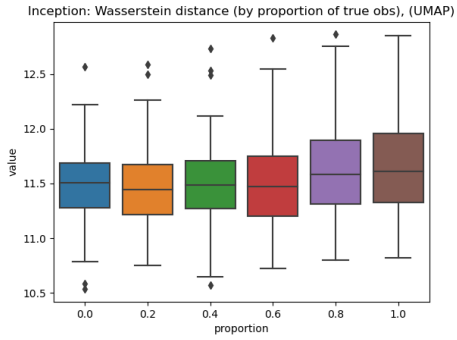
(b) Wassestein, n=250



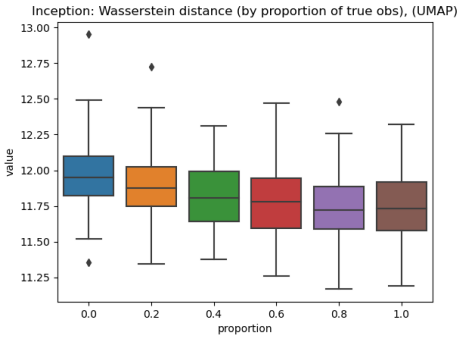
(c) Wasserstein with PCA, n=100



(d) Wasserstein with PCA, n=250



(e) Wasserstein with UMAP, n=100

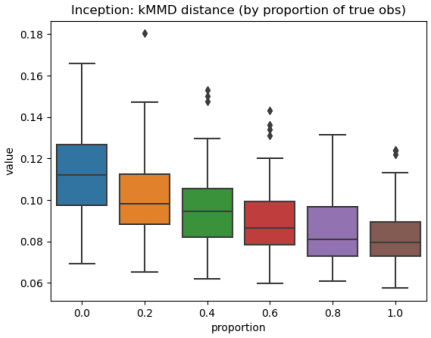


(f) Wasserstein with UMAP, n=250

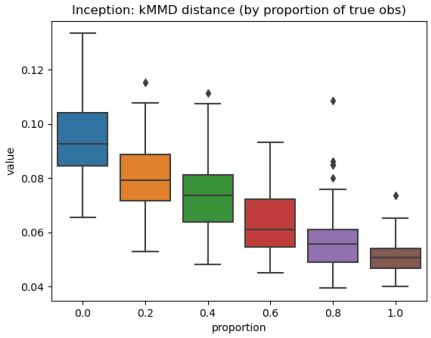
Figure 4.16: Inception, Wasserstein distance (n=100 & 250). Medians similar follow linear trends in full and PCA reduced spaces. UMAP results start faintly showing desired trend at n=250.

In Figure 4.16, we see the results of the Wasserstein distance distributions for sample sizes 100 and 250. We can see linear trends for the full and PCA reduced feature

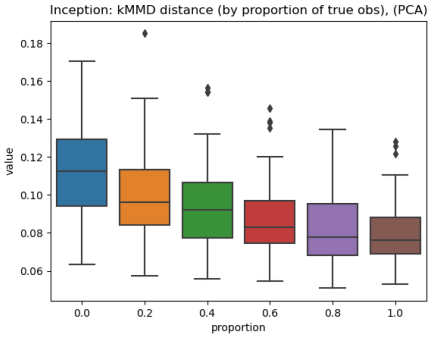
spaces. At n=250, we can see the expected downward trend forming in the UMAP reduced space results. The increase in sample size improves the separation for the other distance measures.



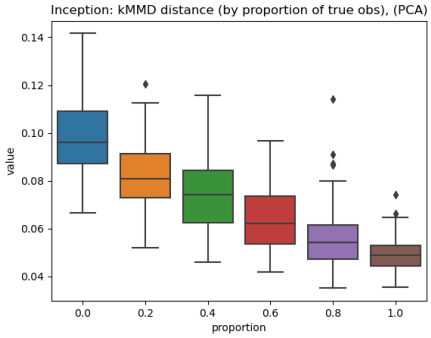
(a) Kernel MMD, n=100



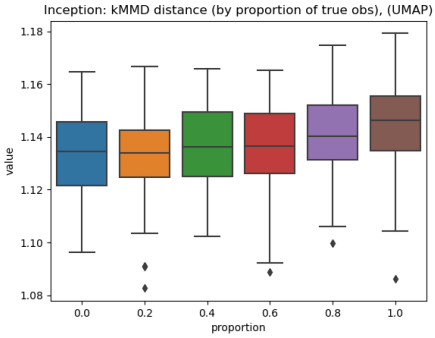
(b) Kernel MMD, n=250



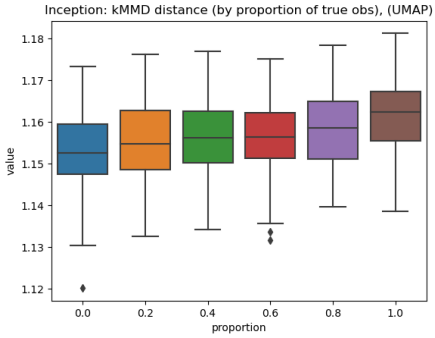
(c) Kernel MMD with PCA, n=100



(d) Kernel MMD with PCA, n=250



(e) Kernel MMD with UMAP, n=100

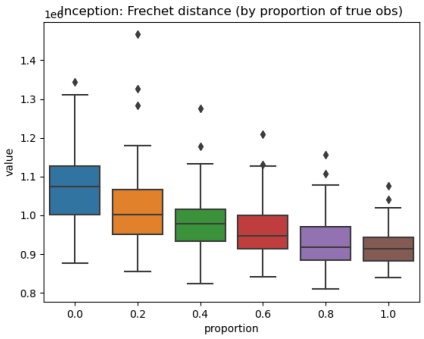


(f) Kernel MMD with UMAP, n=250

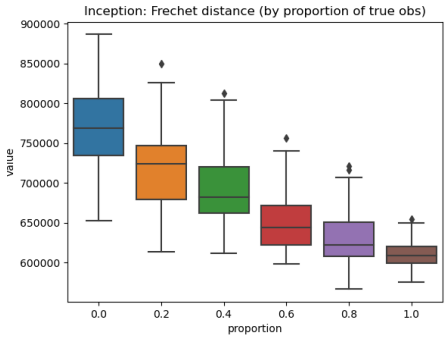
Figure 4.17: Inception, Kernel MMD distance (n=100 & 250). Medians follow somewhat linear trends in full and PCA reduced spaces. UMAP reduced results follow inverse of expected relationship.

In Figure 4.17, we see the results for the Kernel MMD distance distributions. The median distances follow a somewhat linear trend, unlike in the DCGAN feaure space where the trend was exponential. The PCA space reduction has little effect on the behaviours of the distance distributions. We can see that the strange behaviour of the DCGAN UMAP reduced space results is replicated in these Inception space results. The trend is inverse

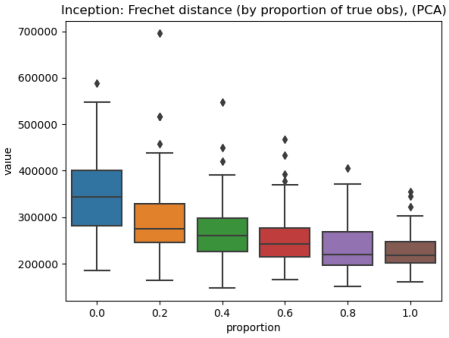
of what we expect. Thus, this behaviour was not caused by the DCGAN feature space, but rather a Kernel MMD and UMAP interaction.



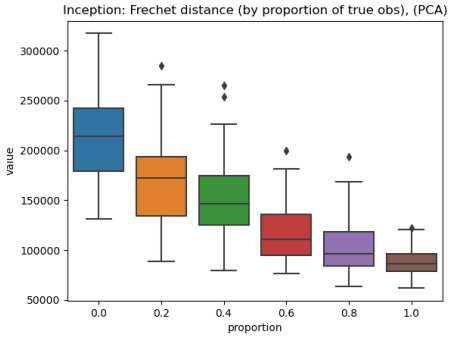
(a) Frechet, n=100



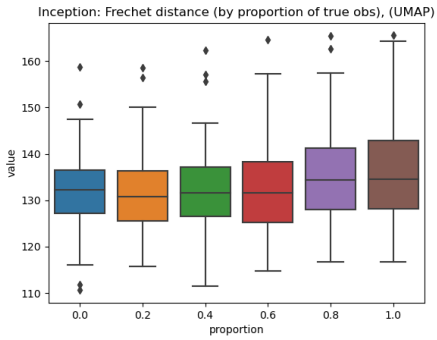
(b) Frechet, n=250



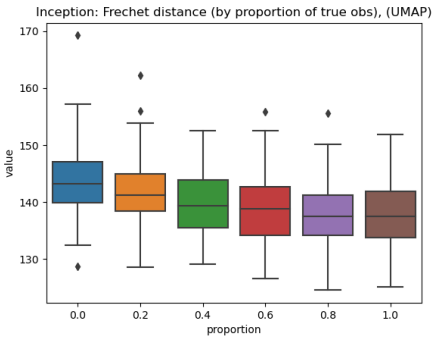
(c) Frechet with PCA, n=100



(d) Frechet with PCA, n=250



(e) Frechet with UMAP, n=100

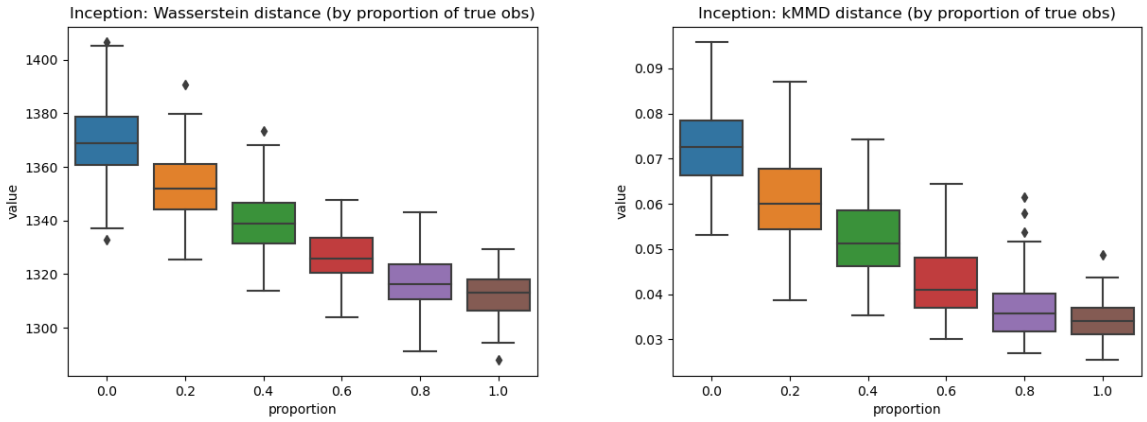


(f) Frechet with UMAP, n=250

Figure 4.18: Inception, Frechet distance (n=100 & 250). Better separation with increase in sample size. PCA reduction has little effect on the trend. UMAP reduction results start exhibiting expected trend at n=250, but only faintly.

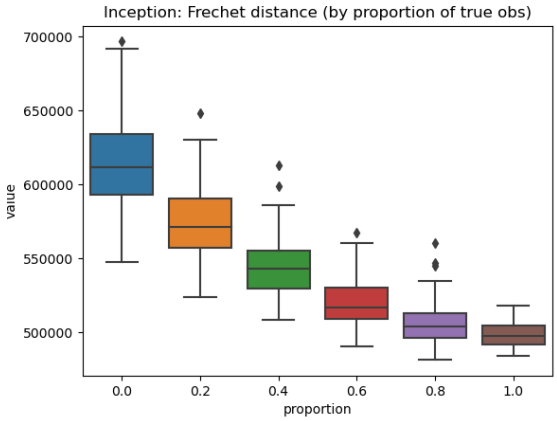
In Figure 4.18 we see the Frechet distance distribution results. The medians follow a somewhat exponential trend. The PCA dimensionality reduction has little effect on the results. The UMAP results start following the expected trend at n=250.

We can see the comparison of the distance distribution for each distance measure, calculated in the full Inception feature space in Figure 4.19. The Wasserstein trend is the most linear, the Frechet trend is the most exponential, whereas the Kernel MMD trend is somewhere in between those two. We can perform the box maximum and minimum comparisons. For the Wasserstein and Frechet distances only the minimum at proportion level 0 is greater than the maximum at proportion level 1. Whereas, for the Frechet distance the minimum at proportion level 0.2 is greater than the maximum at proportion level 1.



(a) Wasserstein

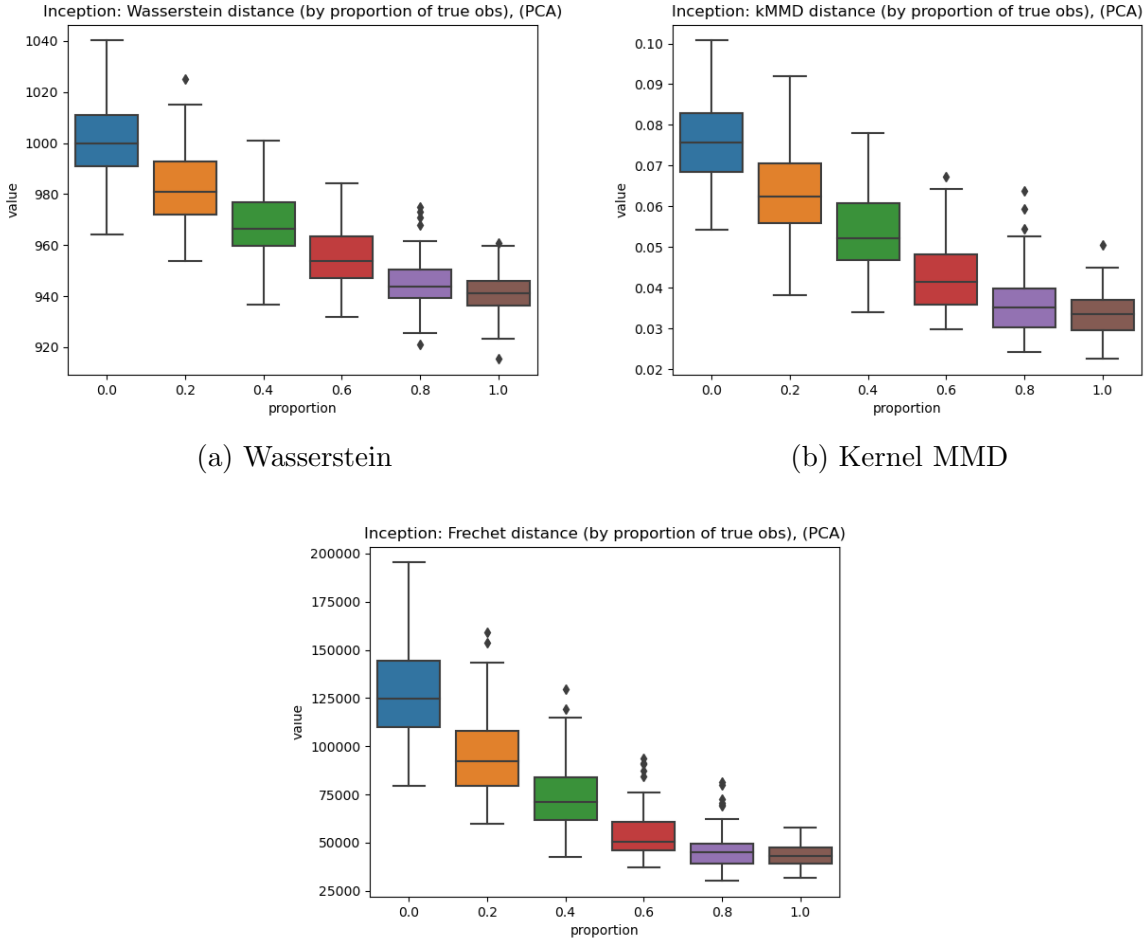
(b) Kernel MMD



(c) Frechet

Figure 4.19: Inception, all distances (n=500). Wasserstein linear trend. Kernel MMD and Frechet somewhat exponential. Distribution ranges increase as proportion value falls.

In Figure 4.20 we see the results of the different distance measures calculated in a PCA reduced Inception feature space. The Wasserstein trend becomes very slightly exponential, the trends of the Kernel MMD and Frechet medians are exponential. The box minimum and maximum test results remain the same as those in the full space. Suggesting that little information has been lost during the PCA dimensionality reduction process.



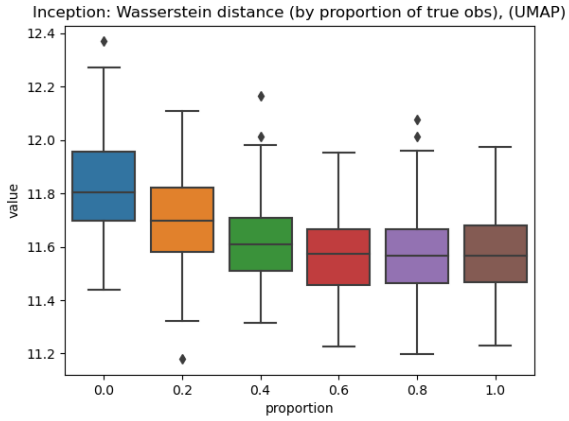
(a) Wasserstein

(b) Kernel MMD

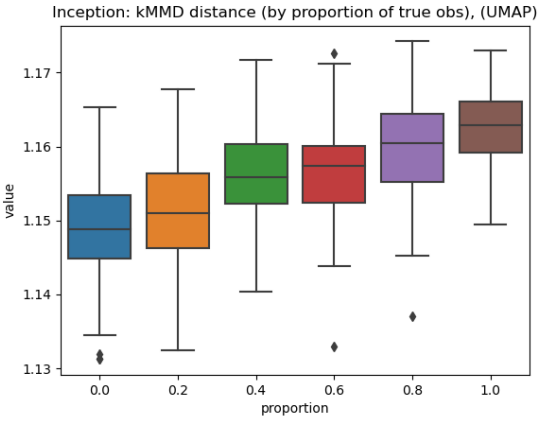
(c) Frechet

Figure 4.20: Inception, PCA reduced, all distances (n=500). Wasserstein trend slightly exponential, Kernel MMD and Frechet trends exponential. For all the distributions, ranges increase as proportion value falls.

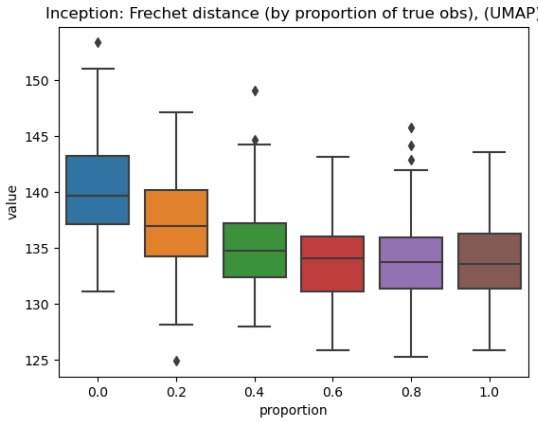
In Figure 4.21 we see the results of the different distance measures calculated in a UMAP reduced Inception feature space. We can see that the Wasserstein and Frechet results follow the expected trend. Although, there is a lot of overlap in the distance distributions. The Kernel MMD results continue to follow the inverse of the expected trend. Compared to the full feature space and PCA reduced space results, these results are quite poor.



(a) Wasserstein



(b) Kernel MMD



(c) Frechet

Figure 4.21: Inception, UMAP reduction, all distances (n=500). Wasserstein and Frechet exhibit expected trend. Kernel MMD trend is inverse to expected trend. Separation worse then the other feature spaces.

4.3.2.3 Overall Comparison

In order to better compare all these results we generated Table 4.2. Where the differences between the distance distributions are displayed in a quantitative manner. The UMAP reduced space results are omitted in the table as they are clearly poor and thus not interesting for fraud detection applications. For each combination of feature space (FS), sample size (SS), dimensionality reduction (DR) method, and distance measure, the percentage change in the average distance was calculated for each proportion level against the average distance at proportion level 1 (when 100% of observations are true). We did consider other way of calculating these difference. Some did not work well when the distance distributions became too different, such as the Kolmogorov-Smirnov test. Others, such as the 1-D Wasserstein distance were too dependent on the scale of the distance results, meaning that normal distributions with means of 100 and 120 would have a greater distance between them than distributions with means of 10 and 12.

Table 4.2 allows us to observe interesting behaviours that we were not able to see by simply looking at the plots. We can see that the average Wasserstein distance increases by very similar amounts across all the possible factor combinations. We can see that the PCA dimensionality reduction benefits the Inception distance differences. This could be explained by the fact that the Inception network is trained on a much larger variety of subjects/classes, meaning that there are activation values that generate unnecessary noise. Whereas, for the DCGAN distance, the PCA dimensionality reduction has little effect on the Wasserstein distances. We can also see that the only decreases in the average distance occur in the DCGAN PCA reduced feature space for the Kernel MMD and Frechet distances.

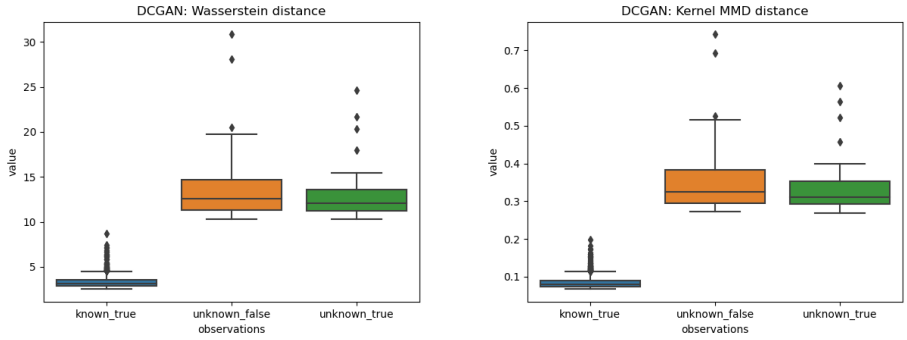
FS	SS	DR	Measure	Percentage of true observations				
				0%	20%	40%	60%	80%
DCGAN	100	none	wasserstein	5.58	4.58	3.06	2.42	1.1
		none	kmmd	26.47	17.13	10.49	5.53	3.27
		none	frechet	20.28	15.76	10.81	7.47	3.28
		PCA	wasserstein	4.58	3.39	2.07	1.61	0.84
		PCA	kmmd	13.07	4.58	2.0	-1.87	0.29
		PCA	frechet	22.2	10.34	5.68	0.48	-0.21
DCGAN	250	none	wasserstein	5.85	4.88	3.51	2.12	0.97
		none	kmmd	52.25	39.25	22.97	10.78	2.46
		none	frechet	26.65	20.68	14.52	8.24	3.57
		PCA	wasserstein	6.0	4.79	3.39	1.9	0.7
		PCA	kmmd	38.07	25.52	11.51	2.74	-1.94
		PCA	frechet	67.18	42.92	24.86	9.64	1.05
DCGAN	500	none	wasserstein	5.98	4.68	3.12	2.16	0.97
		none	kmmd	80.04	53.5	30.32	13.35	4.18
		none	frechet	29.98	22.32	14.2	8.85	3.44
		PCA	wasserstein	6.51	4.86	3.1	1.91	0.73
		PCA	kmmd	64.77	37.32	17.83	0.82	-1.36
		PCA	frechet	**115.36**	69.52	34.76	10.89	-1.43
Inception	100	none	wasserstein	4.53	2.79	1.94	1.22	0.57
		none	kmmd	38.31	22.59	16.16	9.21	3.91
		none	frechet	17.69	10.71	6.7	4.91	1.41
		PCA	wasserstein	6.11	3.54	2.45	1.51	0.73
		PCA	kmmd	43.74	25.77	18.06	9.92	3.95
		PCA	frechet	54.38	30.77	17.85	12.02	2.97
Inception	250	none	wasserstein	5.2	3.77	2.97	1.51	0.78
		none	kmmd	84.68	58.18	46.79	25.56	12.77
		none	frechet	26.45	17.44	13.36	6.6	3.5
		PCA	wasserstein	7.43	5.22	4.14	2.09	1.02
		PCA	kmmd	98.81	68.03	54.96	30.36	15.19
		PCA	frechet	**142.77**	91.1	69.84	32.78	16.84
Inception	500	none	wasserstein	4.38	3.07	2.06	1.09	0.36
		none	kmmd	**111.63**	78.01	52.75	23.62	6.72
		none	frechet	23.43	15.36	9.56	4.51	1.62
		PCA	wasserstein	6.43	4.36	2.9	1.45	0.46
		PCA	kmmd	**125.07**	87.56	59.43	26.1	6.72
		PCA	frechet	**195.6**	**121.31**	73.28	27.13	6.55

Table 4.2: Percentage change in average distance in comparison to average distance with 100% true observations. Negative change in DCGAN PCA reduced space for Kernel MMD and Frechet distance measures.

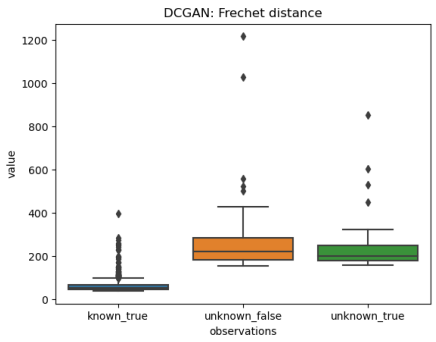
4.3.3 Single Observation Experiment

We now turn to the results from the single observation fraud detection experiment. The experiment was performed in the DCGAN feature space using the same DCGAN model as the multiple observation experiment. We also performed the experiment in the PCA reduced space, the UMAP reduction method was not used due to the poor results it generated in the multiple observation experiment. The number of reference observations to which the distances were calculated were 250. This means that 250 observation were selected from the $known_{true}$ population and the average of the individual distances to those observations was used for the average distance of a new observation. We used 1000 observations from the $known_{true}$ population, non-overlapping with the 250 previously mentioned observations, to build a reference average distance distribution. Finally, we used 100 new observations sampled evenly from the $unknown_{true}$ and $unknown_{false}$ populations. The average distance between each of these 50 true and 50 false observations to the 250 reference observations was then calculated. Since the individual observations are individual samples from multivariate data, we needed to modify them so they became samples of size greater than 1 of multivariate data. We performed the experiment using three different split values: 4, 8, and 16.

4.3.3.1 Full DCGAN space



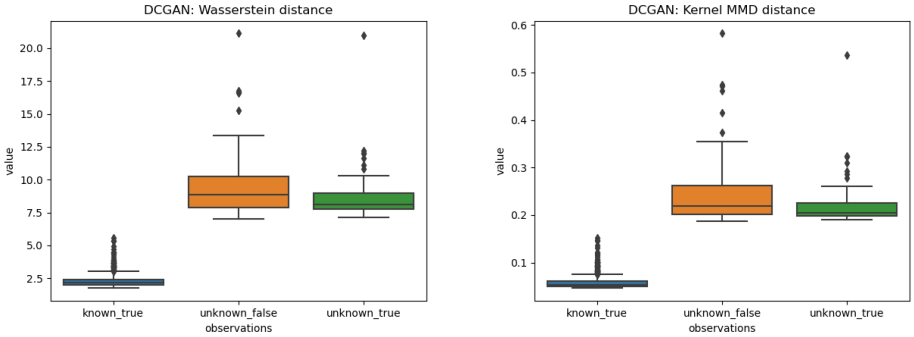
(a) Wasserstein (b) Kernel MMD



(c) Frechet

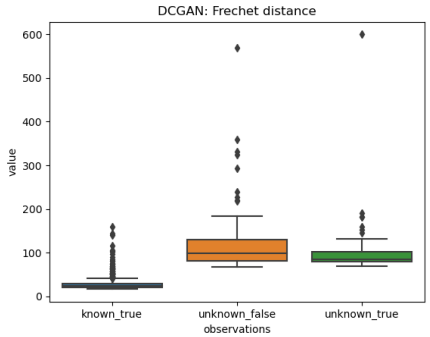
Figure 4.22: DCGAN full feature space, split=4. Known true significantly lower than other two. Unknown true seemingly slightly lower than unknown false. Overlap between outliers and unknown distributions for Frechet distance.

We first look at the results in the full DCGAN discriminator feature space. In Figure 4.22, we can see the average distance distributions for the three types of observations when the split value is 4. We can see that the distribution of the average distances of known true observations is considerable lower than the other two distributions for all distance measures. The behaviours of the unknown distributions, both true and false, are also similar across the distance measures. One could argue that the distribution of unknown true observations is slightly lower than that of the unknown false observations. For the Wasserstein and Kernel MMD distances the lower bounds of the unknown distributions are greater than most of the known distribution outliers, whereas for the Frechet distance this is not the case. For the Frechet distance, some of the known true outliers overlap with the unknown distributions.



(a) Wasserstein

(b) Kernel MMD

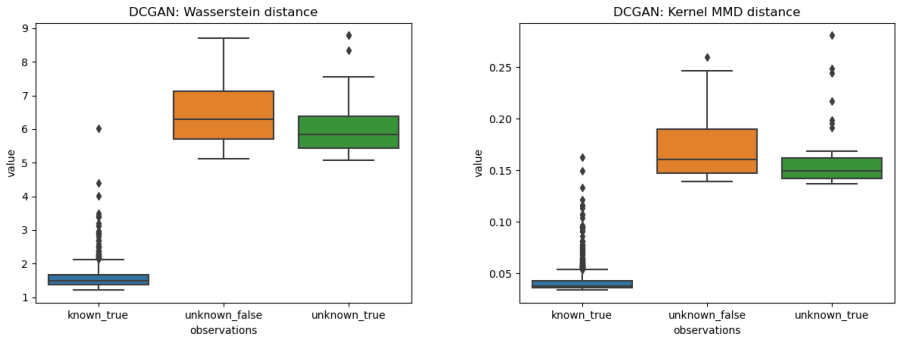


(c) Frechet

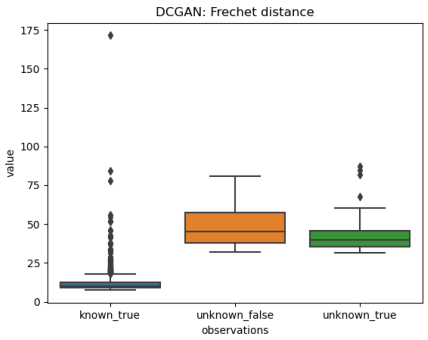
Figure 4.23: DCGAN full feature space, split=8. Known true significantly lower than other two. Unknown true seemingly slightly lower than unknown false. Overlap between outliers and unknown distributions for Frechet distance.

In Figure 4.23, we can see the average distance distributions for the classes of observations when the split value is 8. The behaviour of the average distance distribution of known true observations is the same as when the split value is 4. Again, the unknown true distribution seems lower than the unknown false distribution. Compared to the split value 4 results, it seems that the range of the unknown true distribution is smaller in comparison to the unknown false when the split value is 8. The behaviours are very similar across the three distance measures. Again, there is a difference in the overlap, or lack thereof, between the known true outlier and the unknown distributions. For the Wasserstein and Kernel MMD metrics there is no overlap, whereas for the Frechet distance there is.

In Figure 4.24, we can see the average distance distributions for the three types of observations when the split value is 16. The change in split value seems to again have little effect on the results. The known true distribution is lower than the other two. There is a slight difference between the distributions of the unknown true and unknown false observations, but they overlap significantly. There does not seem be any significant change in the ranges of the distance distributions. The outliers of the known true distribution overlap with the unknown distributions for all three distance metrics, although more so for Frechet than the other two.



(a) Wasserstein (b) Kernel MMD

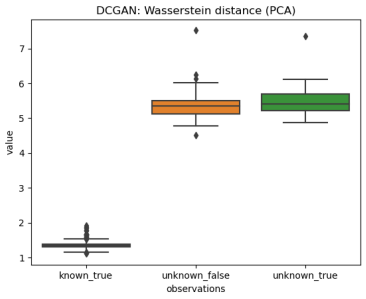


(c) Frechet

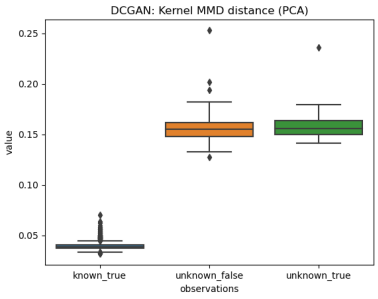
Figure 4.24: DCGAN full feature space, split=16. Known true significantly lower than other two. Unknown true seemingly slightly lower than unknown false. Overlap between outliers and unknown distributions for all three distance measures.

4.3.3.2 PCA reduced DCGAN space

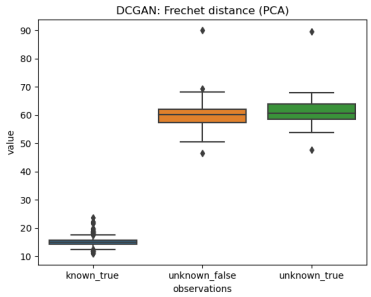
We now turn to the results calculated in the PCA reduced DCGAN discriminator feature space. In Figure 4.25, we can see the average distance distributions for the three types of observations when the split value is 4. The unknown false distance distribution seems to be slightly lower than the unknown true distribution. In the full feature space the opposite was the case, for split value 4. The known true distribution is again significantly lower than the other two distributions. The ranges of the distributions seem to be smaller that in the full feature space. There is no overlap between the known true outliers and the unknown distributions for any of the distance measures.



(a) Wasserstein



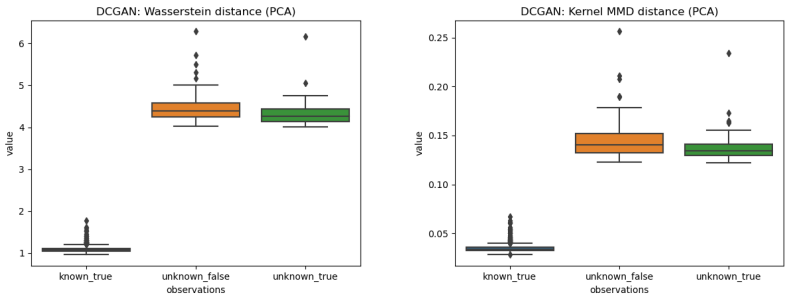
(b) Kernel MMD



(c) Frechet

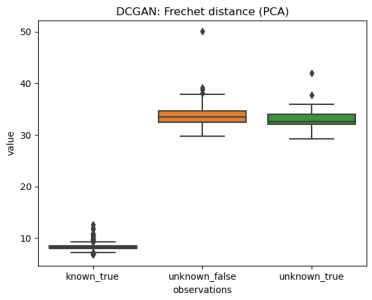
Figure 4.25: PCA reduced DCGAN space, split=4. Known true significantly lower than other two. Unknown true seemingly slightly higher than unknown false.

In Figure 4.26, we can see the average distance distributions for the three types of observations when the split value is 8. Here, the behaviour of the unknown true and unknown false distributions matches the behaviour in the full feature space, the unknown true distribution is seeming lower than the unknown false distribution. The behaviour of the known true distribution persists. Again, compared to the full feature space results for this split value, the ranges of the distributions seem smaller. There is no overlap between the known true outliers and the unknown distributions for any of the distance measures.



(a) Wasserstein

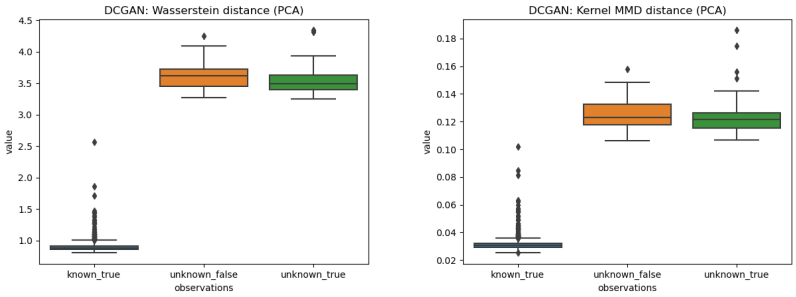
(b) Kernel MMD



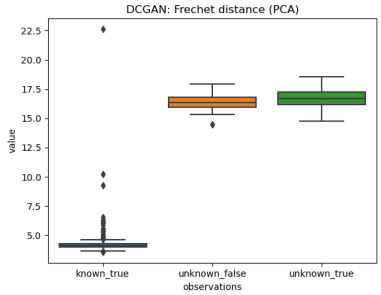
(c) Frechet

Figure 4.26: PCA reduced DCGAN space, split=8. Known true significantly lower than other two. Unknown true seemingly slightly lower than unknown false.

In Figure 4.27, we can see the average distance distributions for the three types of observations when the split value is 16. The same observations can be made about these results as for those when the split value is 8. The known true distribution remains significantly lower than the other two distributions. The unknown true distribution seems to be slight lower than the unknown false distribution, but they overlap significantly. The ranges seem smaller compared to those in the full feature space. The outliers of the known true distribution are closer to the unknown distributions than for the other two split values. For the Frechet distance there even is some overlap.



(a) Wasserstein (b) Kernel MMD



(c) Frechet

Figure 4.27: PCA reduced DCGAN space, split=16. Known true significantly lower than other two. Unknown true seemingly slightly lower than unknown false. Overlap between outliers and unknown distributions for Frechet distance.

4.3.3.3 Single Observation Results Summary

For all factor settings, the average distance distribution for the known true samples was significantly lower than the other two distributions. There are a couple of different possible explanations for this. This distribution includes 1000 observation for which the average distance was calculated, whereas each of the other two distributions only contains 50. It could be that given a similarly large number of samples for the other two distributions, they would become more similar. Another possible explanation is based on the fact that the known true observations were used to train the DCGAN model. If the discriminator 'remembers' these observations, then is is possible that the activation values will be different to observations that the discriminator does not remember. Despite the fact that they belong to the same original distribution or population. If this is the case, then if one would want to generate a reference average distance distribution, one would need a

large enough number of known true observation to train the DCGAN and calculate the reference distance distribution using two disjoint samples.

The results show little difference between the unknown true and unknown false average distance distributions. In the full feature space the unknown true distribution does seem to be lower than the unknown false distribution, but it is not very clear. It would be interesting to see whether this trend persists given more unknown observations. Both distributions are much greater than the known true observation distribution. Thus we cannot use the known distribution to generate the classification threshold. All unknown observations are classified to be false even when the threshold is the 99th percentile of the known distribution. It could be possible to only look at the unknown observations and rank them according to their average distance. Then assign the classes according to the the threshold of this unknown observation distance distribution. However, with these results it is unlikely that the classification performance would be any good as the two distributions have significant overlap.

Overall, the results are expected. We knew that the splitting of an individual observation to create samples of observations was unlikely to work. It is not a method that can be justified in a mathematical or logical sense, rather it was the most efficient way of achieving the goal. The results provide some interesting observations. Mostly the high disparity between the distances for the known true and unknown true observations.

Chapter 5

Discussion and Conclusion

5.1 Discussion

5.1.1 Presented Work

Our first experiment, the feature space experiment, was not particularly informative. There was little difference between the distances calculated in the pixel space versus those calculated in the GAN discriminator feature spaces. We were hoping that the distance between the known and shuffled samples would increase in the feature spaces. It is not clear why exactly the results did not exhibit the expected behaviour. It is possible that the problem lies in the choice of the dataset, or that the shuffled observations contained enough features to cause the distances to be so small.

The multiple observation fraud detection experiment results were very interesting. The first observation we can make is that the GAN discriminator feature space is a viable feature space, as the distance results were similar to those calculated in the Inception feature space. Meaning that the GAN feature space was comparable to the feature space generated by a well known and respected pre-trained classifier network. Thus we can say that discriminator learns features that can then be used for further analysis or modelling.

The multiple observation fraud detection experiment also showed that the GAN evaluation distance measures can be used to distinguish between real and fake observations. The distances between samples of observations increased as the proportion of fake observations in of the samples increased. Exactly the behaviour that we were hoping to see. The distance distributions were noticeably different even when the proportion of fake observations was low. We saw some interesting behaviours from these distance measures in the experiment results. The Wasserstein distance metric is the most indifferent to PCA dimensionality reduction. We saw a linear trend in the distances no matter the feature space. The Kernel MMD and Frechet distances are much more affected by PCA dimensionality reduction. When the proportion of fake observations is low, and the feature space is PCA reduced, they perform poorly and should not be used. However, when the proportion of fake observations is medium to high, they perform well. The Frechet distance sometimes even performs better in such conditions than in the full feature space. We also saw that the UMAP dimensionality reduction method had a negative impact on the results. Despite this negative effect on the results, we still observed an interesting behaviour when the feature space was UMAP reduced. There seems to be an interaction between the Kernel MMD measure and the UMAP reduction. This combination resulted

in the distance distributions following the inverse of the expected pattern.

The single observation fraud detection experiment was unsuccessful. There was little difference between the average distance distribution for the unknown true and unknown false observations. Classification using such distances is unlikely to perform well. The results are not particularly surprising as we were trying to make distance measures, designed to calculate distances between samples of multivariate observations, calculate the distances between individual observations. The method to transform the individual observations into samples of multivariate observations was very crude.

5.1.2 Avenues of Future Work

Despite the results of the feature space experiment, it would still be interesting to investigate the differences between the feature spaces of different GAN models. This could be achieved by including more GAN models in the multiple observation fraud experiment. From the results of this experiment we know that the set-up works, as the results show the expected behaviours for both the GAN model and the Inception model feature spaces.

The multiple observation fraud detection experiment results, raise the question whether there are any applications in which a GAN discriminator feature space could be used. The type of application that seems most suitable would be some sort of unsupervised model. It would be interesting to investigate how well a GAN discriminator feature space performs for other types of data e.g. non-image data. For some types of data, there exist pre-trained networks such as the Inception network that can be used to obtain a feature space, but for some types of data these types of networks do not exist. This is where GAN discriminator features spaces could be beneficial. Of course, the application is limited by the training requirements of GANs, both when it comes to the large dataset necessary, and the computational intensity associated with training. Nonetheless, it could be an interesting avenue for future investigations.

The Kernel MMD and UMAP dimensionality reduction interaction behaviour was an unexpected finding. It would be interesting to see whether this behaviour can be reproduced using different data. If it is reproducible, it could be interesting to investigate why this occurs.

The multiple observation results also inspire a discussion about the training requirement of the GAN discriminator model. It could be interesting to investigate how the distance distributions change as the discriminator is trained. Does the separation and trend of the distance distributions increase as the discriminator is further in the training process, or does further training start being detrimental at some point? This could help better understand what and how the discriminator learns.

The single observation fraud detection experiment could be revisited if new, more sound, methods were available to transform single observations to samples of multiple multivariate observations.

5.2 Conclusion

From our work, we can draw a few interesting conclusions.

We showed that using GAN models for the purpose of obtaining a feature space for the data in question can work. The feature space generated by a GAN discriminator network is a viable option for unsupervised applications. The distances measured in the GAN discriminator feature space were comparable to those calculated in the Inception network feature space, which is a high standard.

We also showed that the GAN evaluation metrics, the Wassertein, Kernel MMD and Frechet distance measures, can be applied to fraud detection. More specifically, fraud detection when assessing reasonably large samples of multivariate observations. All metrics exhibit varying degrees of correlation between the distance distributions and the proportion of fraudulent observations contained within the samples. The strength of the correlation and the trend of the correlation varies depending on the measure used and the feature space in which the distances are calculated. When using PCA dimensionality reduction, and when the proportion of fake observations is low, the Wasserstein metric provides the most consistent and reliable results. If the proportion of fake observations is medium to high, the three metrics provide good results. It is not recommended to use the UMAP dimensionality reduction technique as in the best case scenario the results are poor, and in the worst case scenario, when using Kernel MMD, the lowest distance occurs when all the observations are fake.

Finally, we showed that adapting the data so that the distance measures can be used with single observations does not generate good results. There is little difference in the average distance distributions for unknown true and unknown false samples.

Bibliography

[1] Abadi, Martin et al., 2016. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. pp. 265–283.

[2] Arjovsky, M. and Bottou, L., 2017. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*.

[3] Arjovsky, M., Chintala, S. and Bottou, L., 2017. Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

[4] Arora, S. and Zhang, Y., 2017. Do gans actually learn the distribution? an empirical study. *arXiv preprint arXiv:1706.08224*.

[5] Arora, S., Ge, R., Liang, Y., Ma, T. and Zhang, Y., 2017. Generalization and equilibrium in generative adversarial nets (gans). *arXiv preprint arXiv:1703.00573*.

[6] Barratt, S. and Sharma, R., 2018. A note on the inception score. *arXiv preprint arXiv:1801.01973*.

[7] Borji, A., 2019. Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, 179, pp.41-65.

[8] Chen, J., Shen, Y. and Ali, R., 2018, November. Credit Card Fraud Detection Using Sparse Autoencoder and Generative Adversarial Network. In *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)* (pp. 1054-1059). IEEE.

[9] Chollet, François and others, 2015. Keras. URL: https://keras.io.

[10] Deng, L., 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine, 29*(6), pp.141-142.

[11] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K. and Fei-Fei, L., 2009, June. Imagenet: A large-scale hierarchical image database. In 2009 *IEEE conference on computer vision and pattern recognition* (pp. 248-255). Ieee.

[12] Diggle, P.J. and Gratton, R.J., 1984. Monte Carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society: Series B (Methodological), 46(2)*, pp.193-212.

[13] Doersch, C., 2016. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*.

[14] Flamary, R., Courty, N., Gramfort, A., Alaya, M.Z., Boisbunon, A., Chambon, S., Chapel, L., Corenflos, A., Fatras, K., Fournier, N. and Gautheron, L., 2021. Pot: Python optimal transport. Journal of Machine Learning Research, 22(78), pp.1-8.

[15] Gansner, E.R. and North, S.C., 2000. An open graph visualization system and its applications to software engineering. *Software: practice and experience, 30*(11), pp.1203-1233.

[16] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2014. Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672-2680).

[17] Goodfellow, I., 2016. NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*.

[18] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V. and Courville, A., 2017. Improved training of wasserstein gans. arXiv preprint arXiv:1704.00028.

[19] Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J. and Kern, R., 2020. Array programming with NumPy. *Nature, 585*(7825), pp.357-362.

[20] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B. and Hochreiter, S., 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems* (pp. 6626-6637).

[21] Hinton, G., Srivastava, N. and Swersky, K., 2012. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on, 14(8)*, p.2.

[22] Hotelling, H., 1933. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology, 24*(6), p.417.

[23] Hunter, J.D., 2007. Matplotlib: A 2D graphics environment. *Computing in science & engineering*, 9(03), pp.90-95.

[24] Im, D.J., Ma, H., Taylor, G. and Branson, K., 2018. Quantitatively evaluating GANs with divergences proposed for training. *arXiv preprint arXiv:1803.01045*.

[25] Isola, P., Zhu, J.Y., Zhou, T. and Efros, A.A., 2017. Image-to-image translation with conditional adversarial networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1125-1134).

[26] Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[27] Kingma, D.P. and Welling, M., 2013. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114.

[28] Kolouri, S., Nadjahi, K., Simsekli, U., Badeau, R. and Rohde, G., 2019. Generalized sliced wasserstein distances. In Advances in Neural Information Processing Systems (pp. 261-272).

[29] Krizhevsky, A. and Hinton, G., 2009. Learning multiple layers of features from tiny images.

[30] LeCun, Y., 1998. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

[31] LeCun, Y. and Bengio, Y., 1995. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks, 3361*(10), p.1995.

[32] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), pp.2278-2324.

[33] Mao, X., Li, Q., Xie, H., Lau, R.Y., Wang, Z. and Paul Smolley, S., 2017. Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision* (pp. 2794-2802).

[34] McCulloch, W.S. and Pitts, W., 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4), pp.115-133.

[35] McInnes, L., Healy, J. and Melville, J., 2018. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

[36] McKinney, W., 2010, June. Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51-56).

[37] Nagy, G., 1991. Neural networks-then and now. *IEEE transactions on neural networks*, 2(2), pp.316-318.

[38] Van Oord, A., Kalchbrenner, N. and Kavukcuoglu, K., 2016, June. Pixel recurrent neural networks. In *International Conference on Machine Learning* (pp. 1747-1756). PMLR.

[39] Oord, A.V.D., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A. and Kavukcuoglu, K., 2016. Conditional image generation with pixelcnn decoders. *arXiv preprint arXiv:1606.05328*.

[40] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L. and Desmaison, A., 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, pp.8026-8037.

[41] Python Core Team (2015). Python: A dynamic, open source programming language. Python Software Foundation. URL: https://www.python.org/.

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

[43] Rosenblatt, F., 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review, 65*(6), p.386.

[44] Rumelhart, D.E., Hinton, G.E. and Williams, R.J., 1985. *Learning internal representations by error propagation*. California Univ San Diego La Jolla Inst for Cognitive Science.

[45] Salakhutdinov, R. and Hinton, G., 2009, April. Deep boltzmann machines. In *Artificial intelligence and statistics* (pp. 448-455).

[46] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A. and Chen, X., 2016. Improved techniques for training gans. In *Advances in neural information processing systems* (pp. 2234-2242).

[47] Schreurs, J., De Meulemeester, H., Fanuel, M., De Moor, B. and Suykens, J.A., 2021. Leverage Score Sampling for Complete Mode Coverage in Generative Adversarial Networks. *arXiv preprint arXiv:2104.02373*.

[48] Sinn, M. and Rawat, A., 2018, March. Non-parametric estimation of jensen-shannon divergence in generative adversarial network training. In International Conference on Artificial Intelligence and Statistics (pp. 642-651).

[49] Szegedy, C., Ioffe, S., Vanhoucke, V. and Alemi, A.A., 2017, February. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence.*

[50] Theis, L., Oord, A.V.D. and Bethge, M., 2015. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*.

[51] Villani, C., 2009. *Optimal transport: old and new* (Vol. 338, p. 23). Berlin: Springer.

[52] Waskom, M.L., 2021. Seaborn: statistical data visualization. *Journal of Open Source Software, 6*(60), p.3021.

[53] Xu, Q., Huang, G., Yuan, Y., Guo, C., Sun, Y., Wu, F. and Weinberger, K., 2018. An empirical study on evaluation metrics of generative adversarial networks. *arXiv preprint arXiv:1806.07755.*

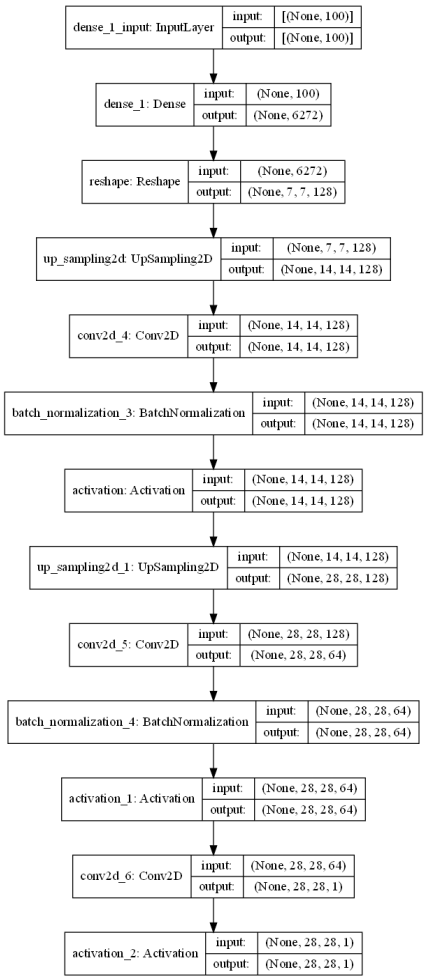
[54] Zheng, P., Yuan, S., Wu, X., Li, J. and Lu, A., 2019, July. One-class adversarial nets for fraud detection. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 33, pp. 1286-1293).

Chapter 6

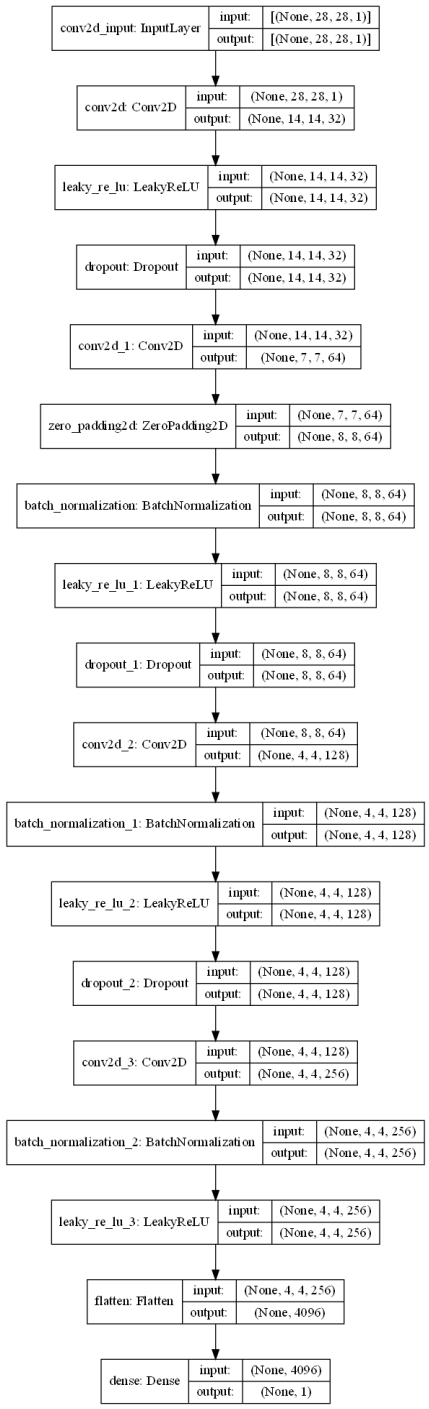
Appendix

6.1 Feature Space Selection Experiment

6.1.1 DCGAN Model Architecture



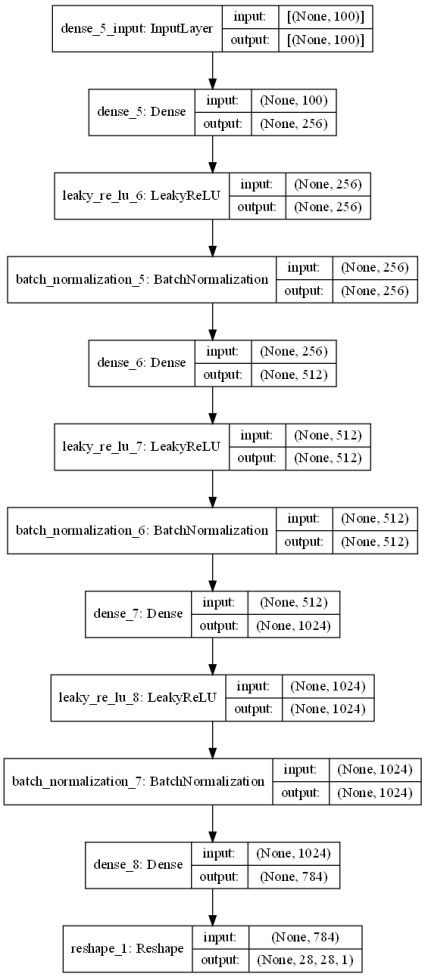
(a) Generator.



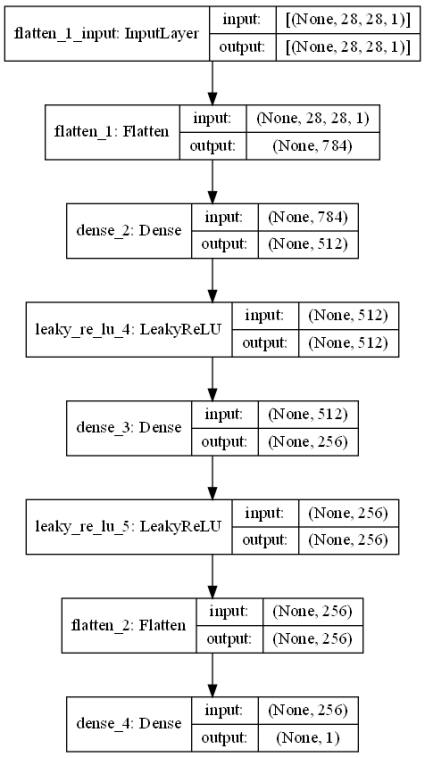
(b) Discriminator.

Figure 6.1: DCGAN architectures.

6.1.2 LSGAN Model Architecture



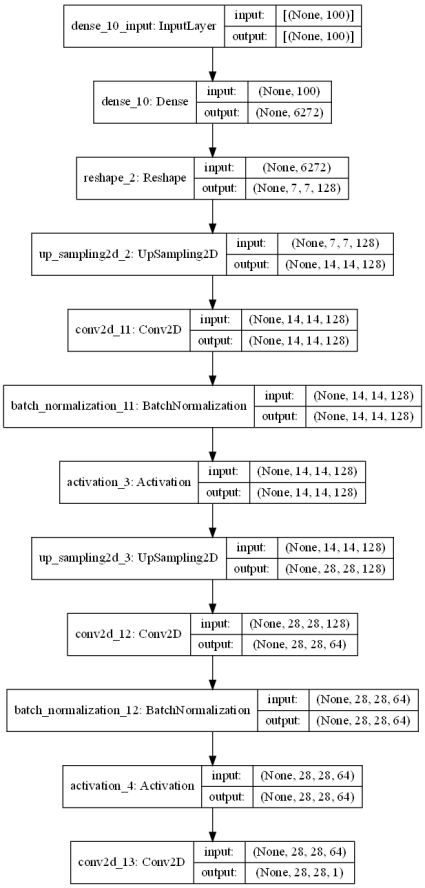
(a) Generator.



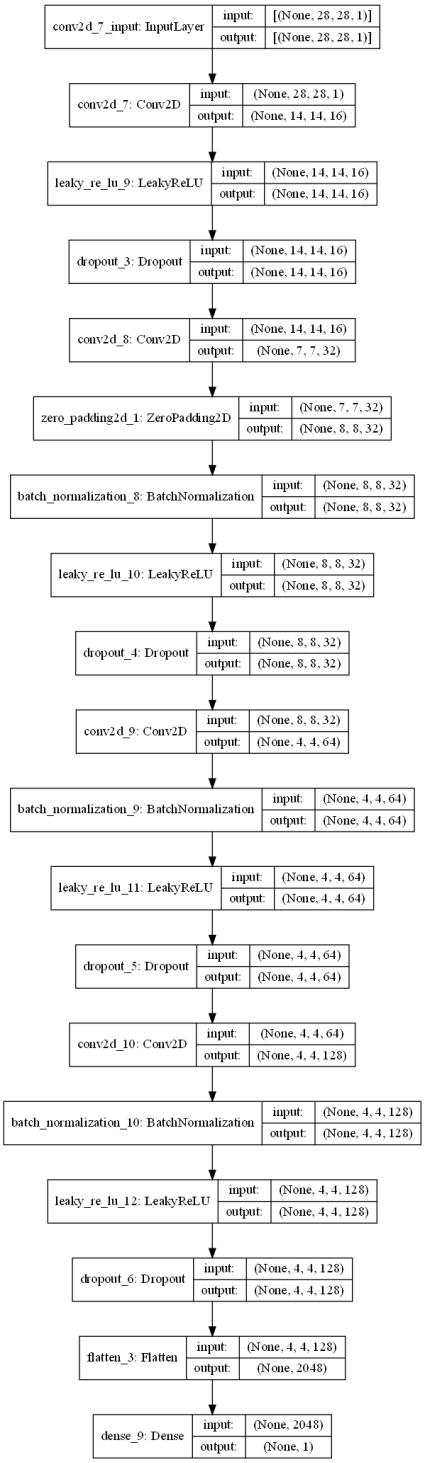
(b) Discriminator.

Figure 6.2: LSGAN architectures.

6.1.3 WGAN Model Architecture



(a) Generator.

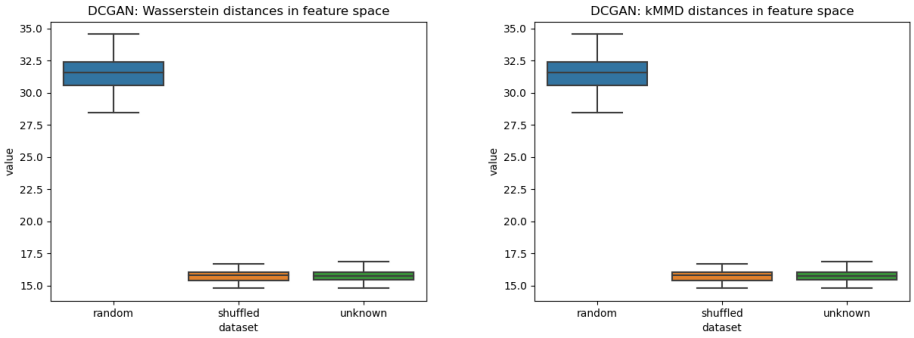


(b) Discriminator.

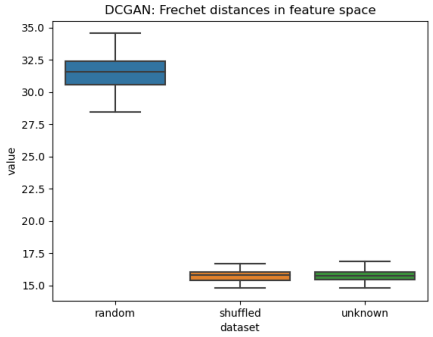
Figure 6.3: WGAN architectures.

6.1.4 Reduced Space Results

6.1.4.1 PCA Reduction

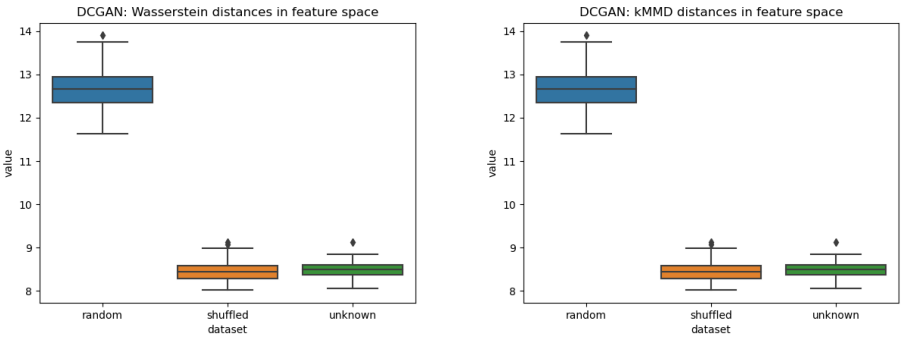


(a) Wasserstein (b) Kernel MMD

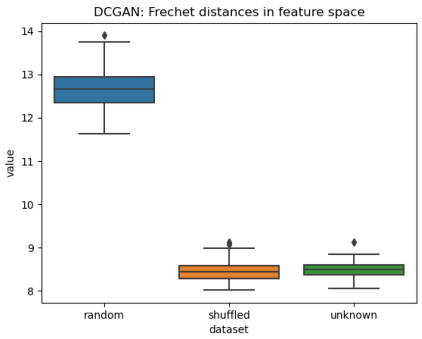


(c) Frechet

Figure 6.4: DCGAN, PCA reduced results.

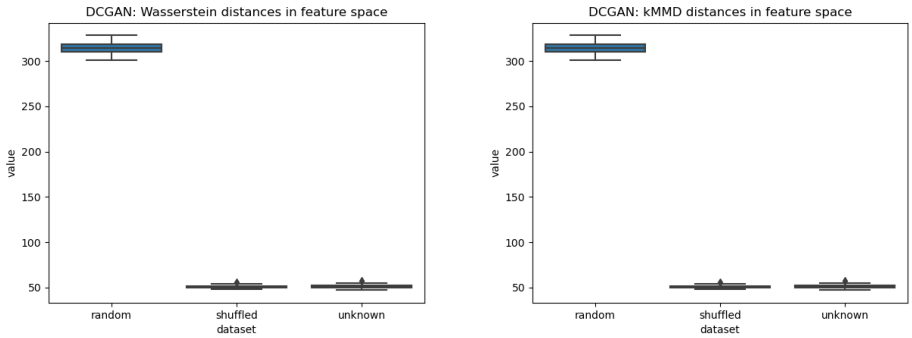


(a) Wasserstein (b) Kernel MMD

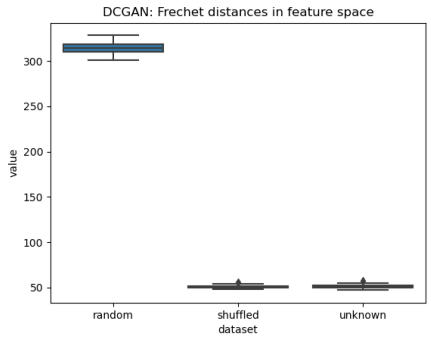


(c) Frechet

Figure 6.5: LSGAN, PCA reduced results.



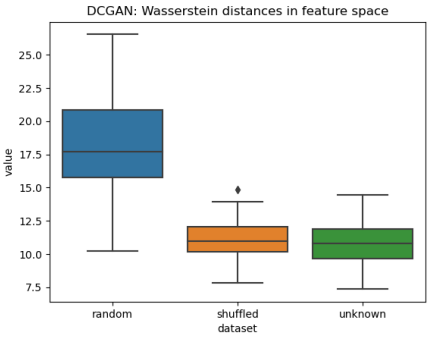
(a) Wasserstein (b) Kernel MMD



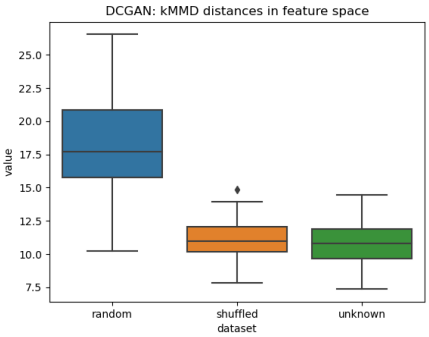
(c) Frechet

Figure 6.6: WGAN, PCA reduced results.

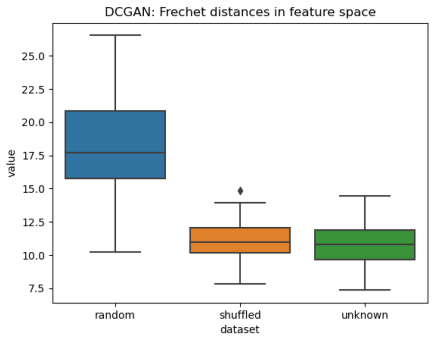
6.1.4.2 UMAP Reduction



(a) Wasserstein

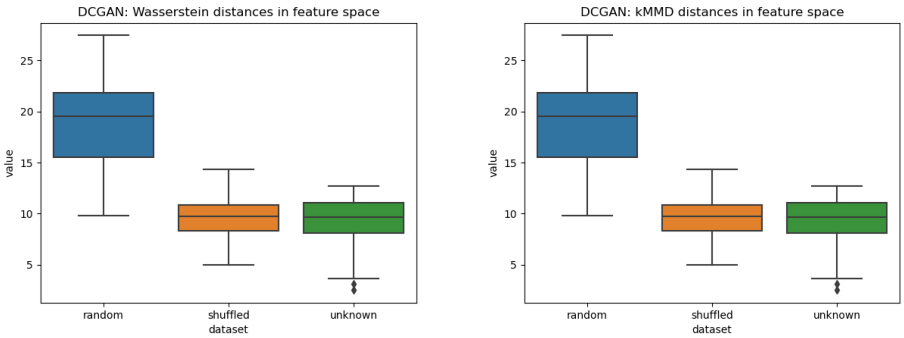


(b) Kernel MMD

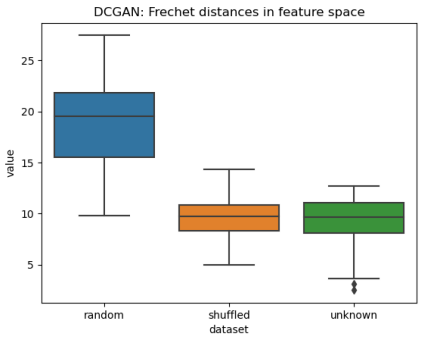


(c) Frechet

Figure 6.7: DCGAN, UMAP reduced results.

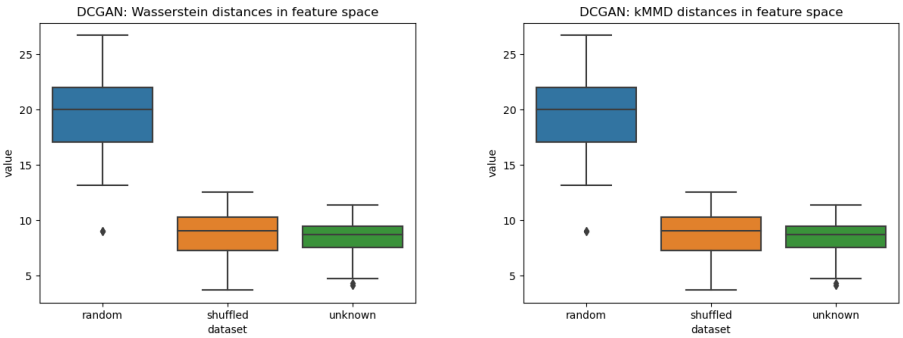


(a) Wasserstein (b) Kernel MMD

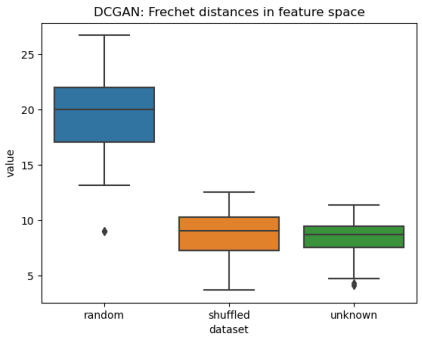


(c) Frechet

Figure 6.8: LSGAN, UMAP reduced results.



(a) Wasserstein (b) Kernel MMD

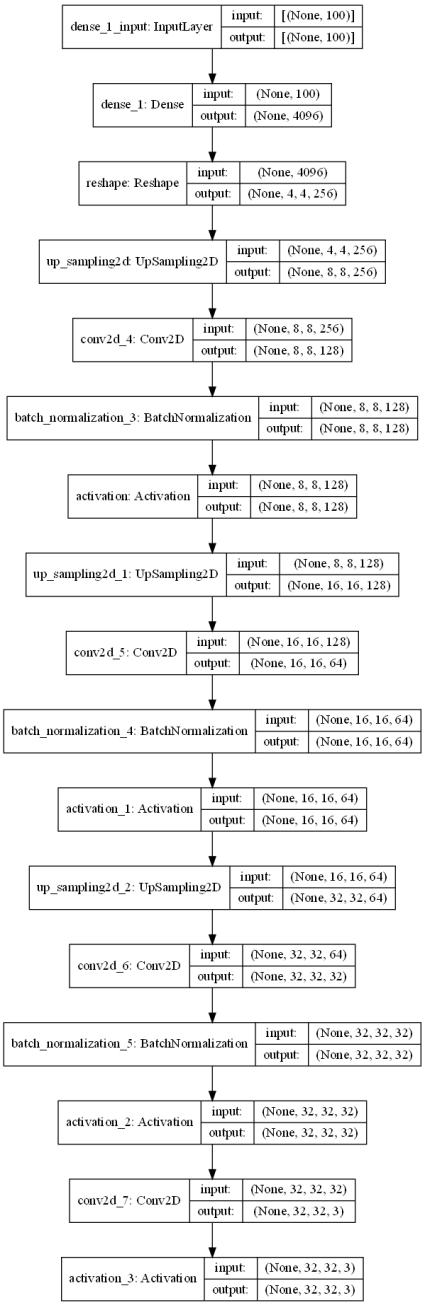


(c) Frechet

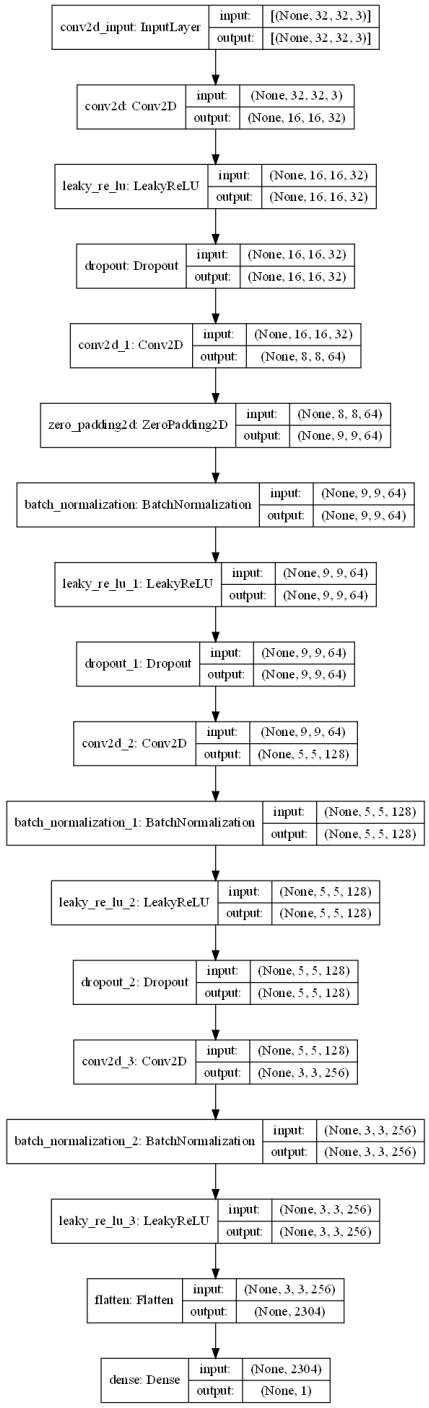
Figure 6.9: WGAN, UMAP reduced results.

6.2 Fraud Detection Experiments

6.2.1 DCGAN Model Architecture



(a) Generator.



(b) Discriminator.

Figure 6.10: DCGAN architectures.

FACULTY OF SCIENCE
DEPARTMENT OF MATHEMATICS
Celestijnenlaan 200B
3001 LEUVEN, BELGIË
tel. + 32 16 32 70 06
fax + 32 16 32 79 98
www.kuleuven.be