



Department of Informatics
King's College London
United Kingdom

7CCSMPRJ Individual Project

3D Generative Adversarial Modelling for Data Augmentation of Human Motions

Name: **Adrian Salazar Gomez**
Student Number: **1870060**
Course: **Data Science**

Supervisor: **Dr Brendan Michael**

This dissertation is submitted for the degree of MSc in **Data Science**.

Abstract

Three dimensional data provides meaningful information that other kind of data cannot provide. The complexity of 3D datasets limits the methodologies that can be used to get useful information from 3D data. Deep learning models are able to manage this type of data,in exchange deep learning requires much data to perform well. Data Augmentation uses existing data to create new data with the ability to improve deep learning methodologies. However, traditional data augmentation methodologies are not useful to improve the classification of 3D data. This research shows that Generative Adversarial has the ability to synthesise data that can be used to improve the performance of 3D classifiers in dataset of all sizes.

Keywords: Machine Learning, Computer Vision, Generative Models, GAN, Deep Learning, Action Recognition, 3D Data, Data Augmentation

Nomenclature

<i>ANNs</i>	Artificial Neural Networks
<i>CNNs</i>	Convolutional Neural Networks
<i>GANs</i>	Generative Adversarial Networks
<i>3D</i>	Three dimensional
$G()$	Generator
$D()$	Discriminator
$\theta^{(D)}$	Discriminator parameters
$\theta^{(G)}$	generator parameters
ω	Parameters in a Artificial Neural Network

Contents

1	Introduction	1
2	Background	2
2.1	Generative Adversarial Networks	2
2.2	Artificial Neural Networks & Deep Learning	4
2.3	Convolutional Neural Networks	5
2.4	Performance Measures: Confusion Matrix and Accuracy	6
2.5	3D Data	7
3	Related Work	9
3.1	3D data	9
3.2	3D objects classification methodologies	10
3.3	Generative Adversarial Networks	11
3.3.1	Data augmentation with generative adversarial networks	14
3.3.2	3D Generative Adversarial Modeling	15
4	Approach	17
4.1	Dataset	18
4.2	Data Pre-Processing	19
4.3	Dataset Split	21
4.4	Action Classification	21
4.5	3D Generation & Data Augmentation	22
4.6	Evaluation	27
5	Results	29
5.1	Qualitative Analysis	29
5.2	Augmentation Size	30
5.3	Overall Results	31
6	Conclusion	34
6.1	Further Research	34
	References	35
	A Table Number corresponding with the classified actions	43
	B Confusion Matrices	43
	C Code	46

List of Figures

1	Traditional structure of a Generative Adversarial Network	2
2	Artificial Neural Network structure	4
3	Deep Learning basic structure	4
4	Stride and filter example	5
5	Convolutional Neural Network structure	5
6	3D convolution	6
7	Confusion Matrix example	6
8	Triangular mesh example	7
9	Triangular mesh example	7
10	Point cloud representation	8
11	6x6x6 voxel grid representation	8
12	Standard 3D data formats	9
13	Conditional GANs (CGANs) standard structure	12
14	ALI/BiGAN basic structure	12
15	Adversarial autoencoder GAN basic structure	13
16	Dynamic FAUST: action as a sequence of frames	18
17	Dynamic FAUST frame samples	18
18	Voxelization process	19
19	Frames of a sequence in different 3D formats	20
20	VoxNet Architecture	22
21	The discriminator in 3D-GANs	23
22	The generator in 3D-GANs	24
23	3D-GAN standard structure	24
24	GANs training evolution whit high discriminator learning rate	25
25	Vanishing Gradient in two GANs training process	26
26	Synthesis quality: Vanishing Gradient	26
27	Data Augmentation Process	27
28	Example of Validation Evolution of 3D Classifier	27
29	Project pipeline	28
30	Real and Synthethic data sample	29
31	Accuracy fluctuation among the proposed augmentation strategies	31
32	Augmented Classifiers vs Standard Classifiers	32
33	Normalised Confusion Matrix of the classifier trained with the small dataset	43
34	Normalised Confusion Matrix of the classifier trained with the augmented small dataset	44
35	Normalised Confusion Matrix of the classifier trained with the medium dataset	44
36	Normalised Confusion Matrix of the classifier trained with the augmented medium dataset	45
37	Normalised Confusion Matrix of the classifier trained with the big dataset	45

38	Normalised Confusion Matrix of the classifier trained with the augmented big dataset	46
----	---	----

List of Tables

1	Accuracy of 3D classifier with multiple augmented sets	30
2	Performance comparison between augmented classifiers and non augmented	32
3	Small Dataset Label Accuracy comparison	33
4	Medium Dataset Label Accuracy comparison	33
5	Full Dataset Label Accuracy comparison	33
6	Label assigned to each action	43

1 Introduction

Understanding three dimensional(3D) information of an object is an important tasks on areas such as computer vision [1], augmented reality [2], virtual reality [3], medicine [4], and robotics [5]. This data provides more information than standard images, particularly, in situations where volume, shape, and motion characteristics play an important role. Although most of the analysis of 3D data require the implementation of machine learning methods, the complexity of 3D representations has limited the application of traditional machine learning. Deep learning methodologies have demonstrated good results in handling 3D data for supervised and unsupervised tasks [6] [7]. However, deep learning models require a large amount of data to perform well and are sensitive to class imbalances. This often proves to be problematic with 3D data as in practical scenarios the amount of data available is limited, particularly, in medicine. Additionally, the collection of 3D data requires special tools such as LiDAR scanner or RGBD cameras.

In computer vision, traditional methods to increase the amount of data available consist on producing small modification of the original data such as image rotation and flipping. These techniques tend to fall short on improving deep learning models performance since the generated variance is minimum. Generative adversarial networks(GANs) are deep learning structures able to learn the distribution of a dataset and synthesise non seen instances with similar characteristics as the original. The data produced by GANs has the potential to introduce enough variation to improve significantly the results of deep learning models, even when data is very limited [8] [9]. With 3D data, traditional augmentation methods do not work well [10] and traditional 3D generation models do not introduce enough variance as are based on mixing part of 3D data to generate new ones [11]. 3D GANs have the ability to learn 3D distribution and synthesise new 3D data. However, to the best of our knowledge, the capacity of 3D GANs to generated data for augmentation purposes has not been tested. This research evaluates the suitability of 3D GANs to augment 3D datasets and improve 3D deep learning methodologies.

To evaluate the capacity of 3D GANs to augment 3D datasets, the research uses a 3D deep learning classifier to identify actions represented in 3D characterisations of humans performing actions. Then analyses the impact of the augmentation process on the classifier. The results confirms the capacity of GANs to improve the performance of 3D deep learning models, even when the data set is limited in size. Additionally, the research evaluates some aspects of the augmentation process that must be considered to maximise the performance of a 3D GANs augmentation process. To our knowledge, this is the first data augmentation strategy suggested for 3D data using 3D GANs.

2 Background

2.1 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a type of machine learning structure proposed in [12] typically used for semi-supervised and supervised task. GANs model the distributions of high dimensional data even when the number of labelled data is scarce. This learnt distributions can be used to synthesise data with similar characteristic as the original data, image processing [13], style transfer [14], data augmentation [8], anomaly detection [15] and classification [16]. Because, the potential of GANs, the literature is continuously proposing new GANs structures that add new functionalities to the original structure. In most of the cases, the experiments made with GANs use images data. However, other types of data such as audio [17], text [18] and graph data [19] have been proposed to its implementation with GANs.

GANs are made of two structures a generator $G()$ and a discriminator model $D()$. The generator $G()$ generate data that comes from the same distribution as the real data. Whereas the discriminator $D()$ differentiates between real data and synthetic data generated by $G()$. The generator $G()$ is a differentiable function that uses a set of parameters $\theta^{(G)}$ to synthesise data by mapping a latent space z inferred from a prior distribution to a sample with the same characteristics as a sample from the real data distribution p_{model} . The generator learns the parameters $\theta^{(G)}$ by feeding synthesised data to the discriminator and learning to fool it. Hence, the generator learns to generate realistic data without any contact with real data x . The typical structure of the generator is a deep artificial neural network model. The discriminator $D()$ is a differentiable function that uses a set of parameters $\theta^{(D)}$ to map an input to a probability of the input being from the same probability distribution as the real data. The input of the discriminator consist on a set of synthesised data $G(z)$ and real data x . The discriminator learns the parameters $\theta^{(D)}$ with a normal supervised learning approach with the goal to label properly real or fake data. Figure 1 illustrates the standard structure of a GAN.

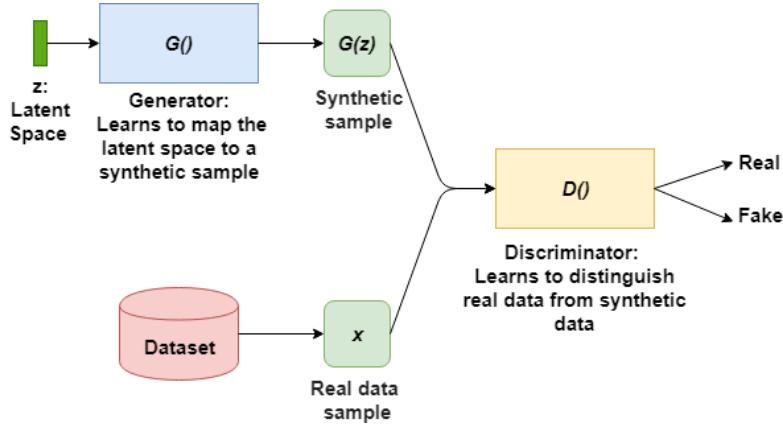


Figure 1: Traditional structure of a Generative Adversarial Network

GANs training procedure is a two-player minimax game where the discriminator tries to maximise the classification performance and the generator tries to minimise the discriminator classification performance. Equation 2.1 is the traditional training objective function in GANs. Whereas equation 2.2 is a modified objective function proposed in [20]. The modified functions produces stronger training signals or gradient to avoid a vanishing gradient situation where $\theta^{(D)}$ and $\theta^{(G)}$ do not change. Section 3.3 contains detailed information about the vanishing gradient problem and the potential solutions suggested in the literature.

$$\min_G \max_D L(D, G) = \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \quad (2.1)$$

$$L(D, G) = \max_D [\mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] + \max_G \mathbb{E}_{z \sim p_z(z)}[\log D(G(z))]] \quad (2.2)$$

Where $\mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$ is the log of the probability of the discriminator of predicting that the generated data is not from the real distribution and $\mathbb{E}_{x \sim p_r(x)}[\log D(x)]$ is the log of the probability of the discriminator to classify real data as real. In the original formulation 2.1, the discriminator parameters $\theta^{(D)}$ are trained by maximising $\log D(x)$ whereas the generator parameters $\theta^{(G)}$ are trained by minising $\log(1 - D(G(z)))$. In equation 2.2 the generator is trained by minimising $\log(1 - D(G(z))$ and the discriminator is trained by maximising $\log D(G(z))$. Initial GANs structures use stochastic gradient descent to update the model parameters. Later structures use the optimisation methodology Adam to update the weights [21]. The updates are made sequentially where either $\theta^{(D)}$ or $\theta^{(G)}$ is updated, while the other parameter is fixed.

Typically, the training stops when the game reaches a Nash equilibrium where one of the players does not change its decision independently of the other player. In most of the cases, there is not a Nash equilibrium and the training stops when the generation does not improve in quality [12]. A generator is said to be optimal when $p_{model} = p(G(x))$ whereas as discriminator is optimal when $D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p(G(x))}$.

One of the biggest problem with GANs is that there is not an standard methodology to measure the performance of the generation process [20]. Thus, complicating the training process. Numerous statistics have been proposed but are designed for particular cases. Section 3.3 covers GANs measures proposed in the literature. This research evaluates the performance of GANs by the increase of accuracy triggered by adding synthetic data to the training set of a classifier.

The popularity of GANs has led to numerous modifications of the original structure. Most of the prominent modifications are Deep Convolutional GANs (DC-GANs), Conditional GANs(C-GANs) [22], Cycle-GANs [14] and Bidirectional GANs [23]. This projects lies heavily on DC-GANs and 3D-GANs. Section 3.3 describes in detail the different types of GANs and their applications.

2.2 Artificial Neural Networks & Deep Learning

An Artificial Neural Network (ANN) is a parametric machine learning model that uses a series of parameters ω to map an input to an output. The basic unit in ANNs are the input layer, hidden layer, and output layer. The input and the output layers represent the input and output of the model respectively. Whereas the hidden layer transforms the input into the output using model parameters. The parameters are learned by using gradient descent or Mean Squared Error Methods. Gradient descent methodologies are more frequent than Mean Squared Errors method. The learning policy tries to minimise a selected cost function given training data. Figure 2 shows a standard ANN structure.

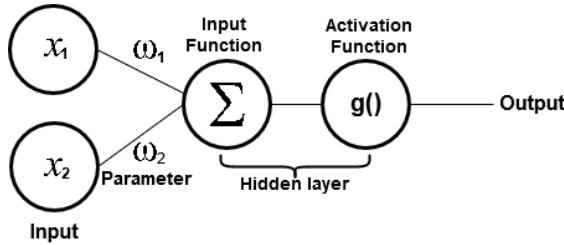


Figure 2: Artificial Neural Network structure

A Deep learning structure is a sequence of ANN layers. This structure can approximate complex non-linear functions. The parameters are trained by using a stochastic back propagation process. This method is a recursive method that transmits the gradient of the last layers to the initial layers of the Deep learning structure. Typically, deep learning structures are feed-forward. A feed forward structure presents the input signal to the network in sequential order without cycles. Some proposed structures contain cycles. Figure 3 represent a simple deep learning structure with three hidden layers.

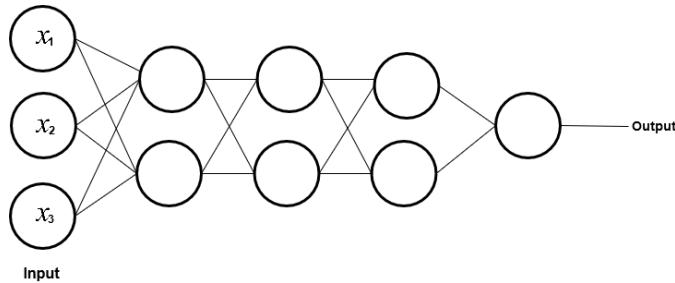


Figure 3: Deep Learning basic structure

There are almost an unlimited number of deep learning structures as there are large number of parameter to combine such as the type of hidden layers, number of hidden layer, types of activation function, and update processes. This leads to the implementation of suggested structures that have demonstrated to perform well on specific tasks. One of the most studied and used structures are Convolutional Neural Networks.

2.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) are deep learning structures frequently used for image classification. These structures reduce complex hierarchical structure, generally images, into a simplified representation that is easier to classify. Then, in most of the CNNs suggested structures, a standard feed forward deep learning network maps the resulting simplified representation into a class. The key elements in a CNNs structure are the convolutional filters, pooling layers, and Rectifier Linear Unit (ReLU) layers.

The key within a CNNs structure are the convolutional filters or Kernels and the strides. The kernels are the window that perform convolution operations over the input. The convolution operation performs a dot product between the network parameters and the model parameter within the window. Then, the output of the dot products are summed up into a value. The network parameters are learned to minimise the loss function of the structure. After each convolution the kernel moves based on the stride. The size of the Kernel window depends on the input size, however, the standard size is 3×3 . Strides indicate the number of steps that the kernel takes after performing one convolution operation. Figure 4 represents a filter of 2×2 size with a stride of 2.

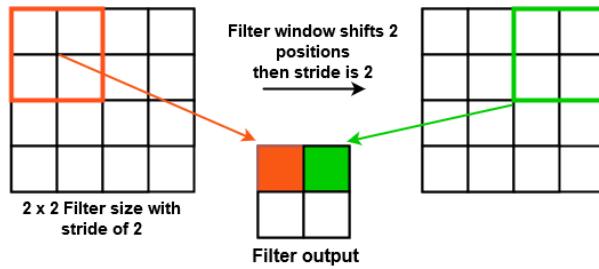


Figure 4: Stride and filter example

Another key element in a CNNs is the pooling layer. Pooling layers reduce the dimensions of the output of a convolutional layer. These layers are filters that parse the entire convolutional layer output keeping the most relevant features for each step of the window. Finally, the Rectifier Linear Unit changes the negative values of the max pooling output to 0. A CNNs consist on a series of convolutional, max pooling and Re-Lu layer that reduces the input size until the resulting input is simple enough to be managed by simple neural networks. Fig 5 illustrates a standard CNNs structure.

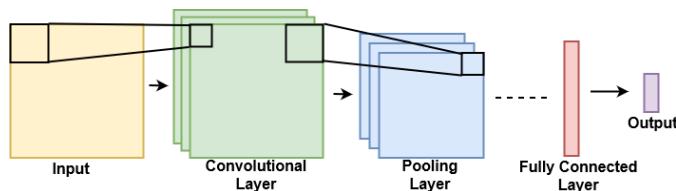


Figure 5: Convolutional Neural Network structure

Although most of the CNNs structures are designed for 2D images, the convolution can also involve a third dimension by adapting the kernel size to include an additional dimension in the convolutions. Figure 6 illustrates a convolution in 3D data. Fig 5

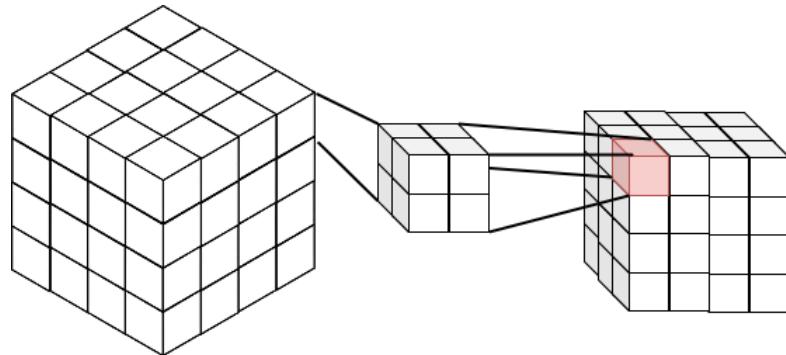


Figure 6: 3D convolution

2.4 Performance Measures: Confusion Matrix and Accuracy

Accuracy and Confusion Matrix are typical evaluation methods for classification methods. Accuracy, formulated in equation 2.3, is the ratio of number of correctly classified instances over the total number of instances.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total Number of predictions made}} \quad (2.3)$$

Confusion matrix evaluates the performance of a classification model on each of the classes. In a confusion matrix representation, typically, each row of the matrix represents the instances in a predicted class while each column represents actual instances in a class. Figure 7 shows the confusion matrix of a binary classifier and the possible outcomes.

		Actual values	
		Positive	Negative
Predicted values	Positive	TP: True Positive	FP: False Positive
	Negative	FN: False Negative	TN: True Negative

Figure 7: Confusion Matrix example

Confusion matrix reports the number of correct and incorrect classifications broken down by class. Whereas Normalised Confusion matrix reports the proportion of correct and incorrect classifications broken down by class. Normalised confusion matrices allow the direct comparison between the individual performance of each class.

2.5 3D Data

This research uses three types of three dimensional data, namely voxelgrids, point clouds and triangular meshes. Triangular meshes represent the surface of 3D objects with a set of triangles that are interconnected by their vertices. Figure 8 illustrates a simple representation of a triangular mesh, complex representations contain more information such as adjacent triangles and edges. Processing triangular meshes can be simplified by doing calculations on the common vertices rather than for each single triangle.

Figure 9 shows an example of a triangular mesh representing a 3D object.

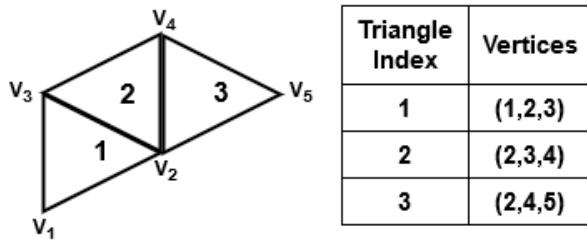


Figure 8: Triangular mesh example

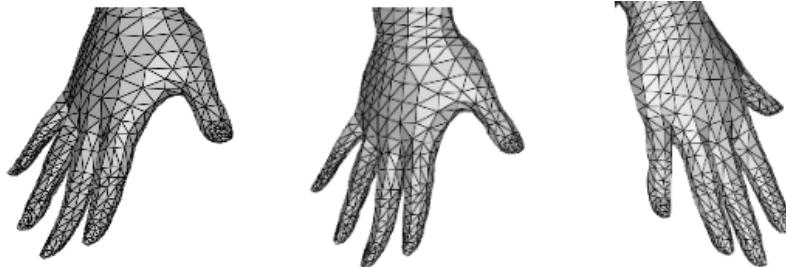


Figure 9: Triangular mesh example

Point clouds represent geometric objects as a set of points in a x,y,z space in a Euclidean coordinate frame [24]. Point clouds are represented as a $N \times 3$ matrix, where N is the number of points. Normally, N is labelled as the point clouds resolution, the higher the number of points used to represent an object the higher the fidelity of the representation. Point clouds are considered a standard format to represent 3D data since they are the output format of common scanning devices such as LIDAR scanners, RGBD cameras, and Kinect [25]. Figure 10 shows a point cloud representation in a x,y,z space.

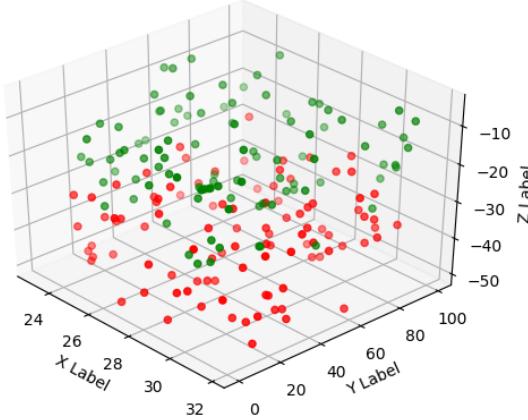


Figure 10: Point cloud representation

A Voxelgrids is a grid in a three dimensional space and a voxels is a point in the three dimensional grid. In contrast to point clouds and triangular meshes, voxels are not represent with x,y,z coordinates. Instead, voxels are represented as a value in a grid that indicates the position of the voxels based upon the other voxels in the grid. The value of a voxels in the grid is usually binary, with 0 indicating that there is not voxel in the coordinate and 1 represents a space occupied by a voxel. Otherwise, to represent voxels in grey-scale, the values in the grid could take values in the (0-1) range. Typically, voxelgrids are represented with 3D arrays. One problem with voxelgrid representations is their sparsity and large dimensionality, a $64 \times 64 \times 64$ array has 264,144 coordinates. Typically, voxelgrids are used in medicine [26] and landscape representations [27]. Sections 3.2 and 3.1 show detailed information about the applications of voxelgrids and methodologies to process them. Figure 11 shows an voxelgrid example.

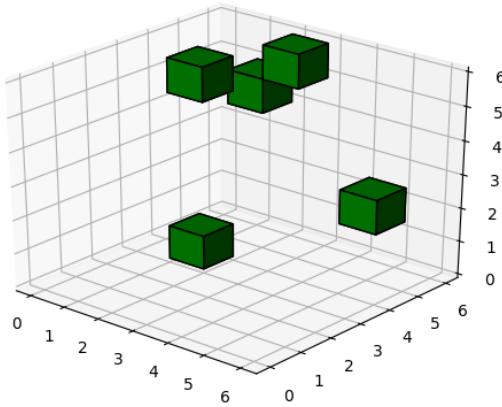


Figure 11: 6x6x6 voxel grid representation

3 Related Work

Identification of human actions has been actively researched in computer vision, however, is a yet under-explored problem because the complexity of modelling human motion. The development of three dimensional representations of real world objects and Generative Adversarial Networks has converged with computer vision resulting in a promising research domain to approach action recognition. This section presents a review of the previous work done in three dimensional computer vision, Generative adversarial networks and its applications in three dimensional computer vision and data augmentation since it is relevant to the work presented in this research.

3.1 3D data

Three dimensional (3D) depictions of objects are a key element in areas such as computer vision [1], augmented reality [2], virtual reality [3], medicine [4], and robotics [5]. 3D data can represent spatial details that are impossible to convey with conventional 2D pictures. The main obstacle to manipulate 3d representation is the high computational and memory cost as a result of the additional dimension [6]. There are multiple formats to represent 3D objects, the most frequent are view-based projections, triangular meshes, volumetric grids, and point clouds. Section 2.5 explains each 3D data format in detail. Figure 12 illustrates the differences between different 3D formats

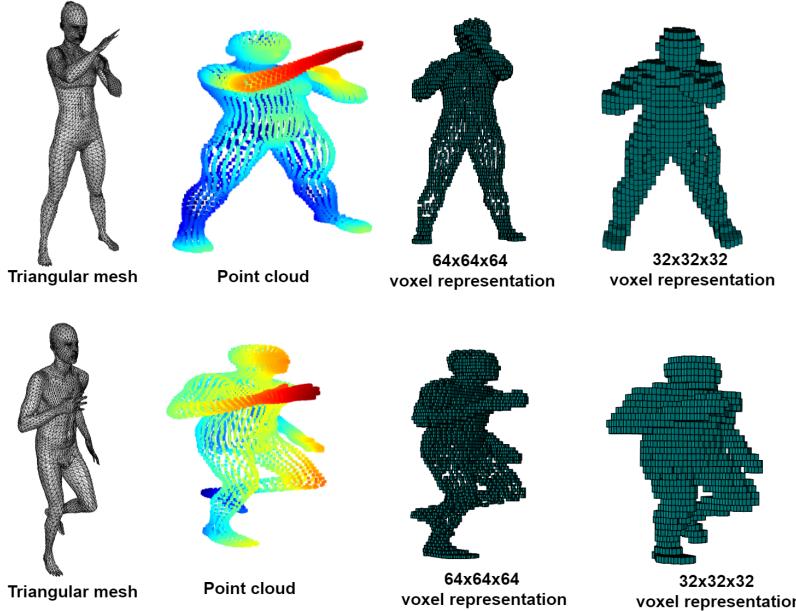


Figure 12: Standard 3D data formats

Multi view projections simplify the analysis of 3D objects but can only show the 3d objects surfaces and does not consider internal information. Triangular meshes and

point clouds encoding format scale better than other formats because its reduced size. Additionally, point clouds are the standard output format of common scanning devices such as LIDAR, RGBD cameras, and Kinect [24]. However, point clouds and triangular meshes do not have uniform dimensions and do not convey any information about the neighbour points of a point and the spaces that are not occupied [25]. Voxelgrids are hard to handle because the large dimensionality but Voxelgrids can convey spatial and neighbour information that other 3D formats cannot convey. Within the computer vision domain, particularly in the shape recognition branch, the focus is in the elaboration of classification structures able to efficiently process and understand 3D data. Then, implement these classifiers to improve processes in fields where 3D data is frequent.

3.2 3D objects classification methodologies

The large dimensionality of 3D data condition the methodologies that can be used to perform classification in this format. Convolutional neural networks (CNNs) tend to perform better than other methods on big dimensional data which makes CNNs the preferred methodology to classify 3D data. Another approach to classify 3D data is based on the simplification of 3D data into a reduced space that can be used by standard classifiers. Following this approach De Deuge et al. [28] uses unsupervised Deep learning to reduce the data dimensionality, and then applies a nonlinear SVM in the reduced space. Shape descriptors can be extracted from 3D data and then feed a fully connected neural network with the descriptor [29]. Generally, the classification methodologies based on the simplification of 3D data, frequently, do not scale well on large dataset and are slower than methods based on 3D CNNs [6]. On the other side, within the 3D CNNs classification methodologies, there are several approaches that depends on the input format; volumetric CNNs, multi-view CNNs, point cloud CNNs, and spectral CNNs.

Volumetric CNNs use voxelized shapes as input. Voxelized representations are able to provide neighbour information between the elements in the 3D space and distinguish between free and occupied spaces [6]. However, Volumetric CNNs are limited by the computational cost of handling big dimensional and sparse high resolution 3d voxelized data [7]. As a result, multiple 3D CNNs deep learning structures have been proposed to make convolutional processes tractable and improve its performance. Shapenet [30] and Voxnet [6] are pioneer 3D volumetric CNNs structures to perform shape classification. Other applications of volumetric CNNs include generative models [31] and variational autoencoders [32]. Volumetric CNNs have been also used for video classification where the third dimension is the time dimension instead of volume [33].

Multiview CNNs transform 3D images into multiple 2D images. Then standard 2D CNNs are implemented for their classification [34] [35]. This approach avoids the high computational cost and memory limitations of 3D data. Multiview CNNs performance relies on 2D CNNs structures, the method to render 3D images into 2D, and the methodology to combine multiple classification result into a single classification. FusionNet [36] ensembles volumetric and multiview CNNs to boost the performance of 3D classifiers.

Spectral CNNs can classify 3D data given in mesh format. Theses methodologies are

limited to manifold 3D meshes and it is no clear how this methodology can be applied to non-isometric meshes. [37] use spectral CNNs to classify 2D data projected into a 3D manifold while [38] use the methodology to classify 3D human shapes.

Point cloud CNNs use point cloud volumetric data to perform classification. Most of the 3D data extraction methodologies produce point clouds by default. Therefore, no data processing is required which avoids loss of information. Additionally, voxel grids or multi-view data are highly voluminous data representations that might result on a computational intractability. One of the difficulties in the development of point clouds classifiers is the unordered structure of the point clouds. Because the unordered structure of point clouds, the classification models must be invariant to the input feeding order and have to be able to capture the relationship between the unordered points. Some networks have been proposed to perform classification with point clouds. Kd-network [39] represents point cloud information with kd-trees that are the input of a CNN structure. Point net[7] is a CNN structure that admits point cloud inputs and PointNet++ [40] is a variation of Point net that creates a hierarchical structure of point clouds where Point net is applied recursively on each of the local structures.

3.3 Generative Adversarial Networks

Generative adversarial networks (GANs) are deep learning generative models. Proposed in [12], GANs are able to model high dimensional data distributions by employing two deep learning structures, namely the generator and the discriminator. The discriminator differentiates between synthesised data and real data while the generator tries to fool the discriminator with synthesised data. Section 2.1 covers technical details of GANs. Among multiple applications, GANs have been used to study the representation and manipulation of data distributions, improve machine learning methodologies, deal with missing or incomplete data, outlier detection and synthesis of realist images [41].

There are five major GANs architectures; fully connected GANs, Convolutional GANs, Conditional GANs, Inference GANs, and adversarial autoencoders [42]. Each of these architectures share the same basic adversarial mechanisms but with structural changes and different functionalities.

Fully connected GANs (Figure 1) are a primitive GANs structure presented in [12] where the generator and the discriminator use fully connected networks. This structure is limited to the generation of simple data. Convolutional GANs modify the structure of fully connected GANs with Convolutional neural networks (CNNs) taking advantage of the suitability of CNNs to handle complex images [43]. Wu et al. [31] extends the concept of convolutional GANs to the 3D data domain by using 3D CNNs. 3D GANs are covered in detail in section 3.3.2. Conditional GANs (CGANs), represented in figure 13, were suggested in [22] where the generator and the discriminator are class conditional. CGANs improve the generation of multi-modal data and allows the synthesis of a particular class.

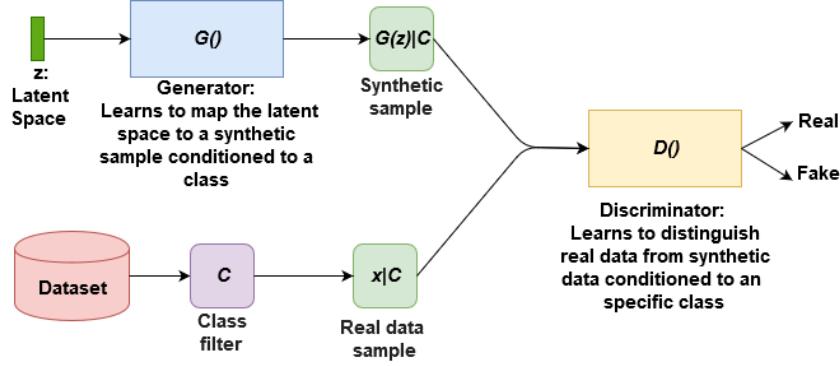


Figure 13: Conditional GANs (CGANs) standard structure

GANs with inference models include an inference mechanism to map real data to the latent space z . The inference system provides GANs with the ability to perform conditional generation, semi-supervised learning and sample reconstruction [16] [23]. Figure 14 illustrates and ALI or BiGAN structure, a standard inference GANs. Illustrated in figure 15, adversarial autoencoders employ an autoencoder in the standard GANs architecture. The autoencoders encoder output aims to match the distribution of the latent space z whereas the autoencoders decoder tries to reconstruct the original image from the encoder's output. In this framework, the discriminator differentiates between GANs latent spaces distributions and the output of the encoder. Adversarial Autoencoders have applications in clustering and semi supervised and supervised learning [44].

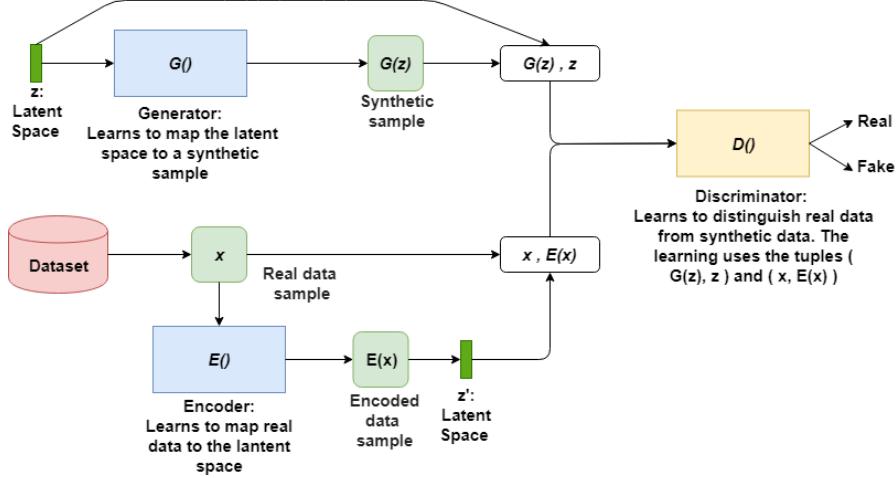


Figure 14: ALI/BiGAN basic structure

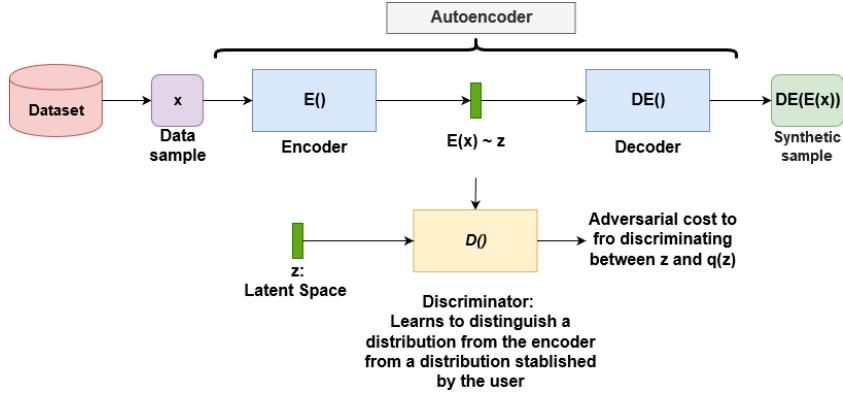


Figure 15: Adversarial autoencoder GAN basic structure

Although image synthesis is a frequent application of GANs [43] [41], GANs can be applied for other tasks such as data augmentation [8], improving performance of reinforcement learning models [45] [46], inference [16] [23], semi-supervised learning [47], imitate agent policies [48], data privacy [10], anomaly detection [49], and domain transfer [50]. Section 3.3.1 covers the application of GANs for data augmentation.

Despite the great success of GANs, GANs training process is unstable and challenging [51]. GANs training is based on the zero-sum non-cooperative game that converges in a Nash equilibrium when one of the players does not change its decision independently of the other player decision. This is the optimal point in the GANs minimax objective function represented in equation 2.2. However, this equilibrium is not guaranteed [20]. Even when the model converges, there is not guaranty that the distribution of the generated data is close to the probability distribution of the original data [52].

Ideally GANs can represent all the distributions within a data set. However, one common problem during the GANs training is 'mode collapse' where the generator only synthesises the same family of samples or just a single type of sample to easily fool the discriminator [51]. Mode collapse arises from situations where the generator is trained extensively without updating the discriminator. Then, the generator finds the data that best fool the discriminator. The diversity of the generator can be improved by using multiple GANs to cover all the modes of the distribution [53]. Vanishing gradient is another common problem during GANs training process where the discriminator loss converges suddenly to zero and the model stops learning [51]. This problem is usually triggered by the discriminator learning faster than the generator. Then, the distributions $p(x)$ and $p(G(z))$ do not overlap and the discriminator can differentiate between them easily. Because the generator is trained via the discriminator, the generator does not receive gradient updates when the discriminator loss converges to 0. Adding noise to the generator have a positive effect on avoiding this problem [51].

Some solutions to improve GANs training relies on modifying the generator and discriminator structure [43], adding noise to the discriminator [54], limiting the discriminator training if its accuracy is under a specific threshold accuracy [31], and modifying

the generator and discriminator cost functions [55] [56]. In addition, Salimans et al. [20] suggests several approaches to improve GANs training process. The first method changes the generator objective to match the generated images with the discriminator’s intermediate activation of the real data. This modification aims to increase the amount of information available. A second methodology, heuristic averaging, aims to speed up model convergence. Heuristic averaging consist on penalising the network weights if these weights deviate from the running average of previous weights. The third, mini-batch discriminator enables the discriminator to be aware of the differences between the generated distribution and the real distribution as a whole. The method compares the distance between batches of real data and synthesised data. Then, the extra feature is used as input for the discriminator to avoid mode collapse. A fourth methodology, one-sided label smoothing establishes the discriminator target for real data as 0.9 instead of 1 to smooth the discriminator decisions and prevent an overconfident discriminator. The fifth approach, virtual batch reduces the dependency of an instance to other instances by normalising every instance within a training mini-batch. The normalisation is based on the statistics of a reference batch retrieved at the beginning of the training process.

While much progress has been made to understand and improve GANs training process[51] [20] [52], there still remain the challenge of measuring GANs performance. There is not an effective methodology to evaluate quantitatively the fidelity of the synthetic images and it is not clear whether different GANs methodologies should be compared [41]. Some used evaluation methodologies are the like-hood estimation [12], and human inspections [57]. The absence of a procedure to measure the generation quality complicates the hyper-parameter tuning process. This is particularly concerning because GANs sensitivity to hyper-parameters [58].

3.3.1 Data augmentation with generative adversarial networks

Data Augmentation is a promising application of GANs aiming to solve problems experienced by deep learning models when the dataset is not big enough. Deep learning models have demonstrated unprecedented performance on machine learning tasks. In exchange, these models require large amounts of data to avoid overfitting and lack of generalisation. Additionally, imbalanced datasets result on the model to fall short.

The literature has developed several techniques to avoid loosing performance because overfitting. One approach is to add additional processes to existing deep learning structures such as batch normalisation [59], normalisation layers [60] and dropout [61]. When the training data is particularly small , these techniques cannot capture properly input invariances that are useful for the training process [8]. Another approach is to generate additional data by modifying the original data with augmentation processes.

Augmentation methodologies apply transformations to the original dataset to create new data and improve the generalisation ability of the classifiers. Common augmentation techniques in 2D and 3D computer vision are flips, rotations, gaussian noise, and random translations [62] [63] [6]. The application of these techniques is a common practise for

large and small datasets because the proven benefits [64]. However, normal augmentation techniques does not represent the underlying data distributions, are limited to simple data variances, and produce highly correlated training data [10]. These limitations motivated the implementation of image synthesis methods able to induce variability to the augmented data while representing the underlying data distributions.

GANs can model wide large invariance and produce data that comes from the original data distribution. Consequently, the literature has started to test the ability of GANs as a method for data augmentation purposes. [8] proposed Data Augmentation Generative Adversarial Network (DAGAN), a GANs framework based on conditional GANs able to transform data withing the same domain. DAGAN transforms data that belong to a class into data of the desired class. DAGAN is implemented to synthesise data that belongs to a class with low frequency to balance the dataset and increase the classification performance on multiple 2D image public datasets. [9] augments the images of a dataset with standard augmentation methods and then synthesises new data with GANs using the already augmented data as a input to improve the liver lesion classification. [10] synthesises brain tumor MRI scans with image-to-image GANs that modify the characteristics of the original images to obtain new images. In addition, This research proves the capacity of GANs to create anonymous synthetic data to be used to train effective classification and segmentation methods. [65] achieves a superior bone lesion classifier by using synthetic data from GANs. To do so, the research uses cycle GANs to synthesise images with bone lesions from a particular part of the body from images without lesions using images with bone lesion from a different part of the body. Finally,[66] suggest Conditional Progressive Growing of GANs (CPGGANs) to synthesise MRI images of brain images with bounding boxes indicating brain metastases to improve the performance of object detection classifiers such as YOLO [67] or R-CNNs [68].

Data augmentation with GANs is particularly suitable in medicine related tasks because the lack of labelled data and the strict privacy requirements. To the best of our knowledge,there is not a proposed GANs data augmentation experiment that uses 3D data and implements a specialised 3D GANs structure for the augmentation process.

3.3.2 3D Generative Adversarial Modeling

Initial GANs architectures work only with 2D data such as images. The increasing popularity of 3D data and the development of 3D deep learning structures instigated the development of 3D GANs structures [31]. 3D GANs make possible to obtains the benefits of using GANs in domains where 3D data is used extensively.

Initial 3D generative methods reconstruct and generate new 3D images with non-parametric approaches based on retrieving and combining elements from the dataset [11] [69]. With this approach, 3D synthesis was constrained by the availability of morphological 3D templates, supervision during the process, and the 3D elements available in the repository [31]. Most of these methods use 3D data formats that can be repre-

sented in 2D such as CAD wire-frames [70], meshes and skeletons [71]. Another 3D image synthesis approach is based on deep learning methods such as Recurrent Neural Networks [72], Deep Belief Networks [30], Deep Convolutional Auto-encoders [73], and Capsule Networks [74]. Using 3D deep learning synthesis methods, [75] [76] synthesise 3D data from 2D data, [72] reconstructs 3D images, [77] simplifies 3D images into discriminative representation, and [78] transforms the 3D data format from point clouds to voxelgrids. Voxelgrids and point clouds are typical 3D data formats used in 3D deep learning synthesising methods. 3D image synthesis with deep learning requires, in most of the cases, full supervision and are limited by the variance that can synthesise.

3D GANs architectures aim to overcome the problems of previous generative methods. Implementations of 3D GANs claim that GANs, in contrast of primitive 3D synthesis methods and other deep learning generation methods, does not require structural templates, does not borrow items from the dataset, generate realistic object with variations, and does not require supervision [31]. However, 3D GANs are particularly hard to train because the big size and complex distributions of 3D data [79]. Depending on the type of 3D data format used as an input for GANs, there are two approaches; GANs that works with voxel grids and GANs that use point cloud 3D data.

Motivated by the lack of GANs methodologies for data in 3D formats, Wu et al. [31] suggested a 3D GANs framework based on volumetric CNNs able to synthesise voxelgrid 3D shapes. Besides synthesis, 3D GANs, once trained, can map complex 3D voxelgrids into an informative feature representation which are able to improve the classification processes. [79] identifies the complex training process of voxelgrids based 3D GANs. As a result, the research proposes a 3D GANs structure to make improvements in training and convergence time. This simplified structure uses a reduced size voxelgrid input and a training objective function based on the Wasserstein distance with gradient normalisation [55]. Voxelgrid based 3D GANs have been applied for 3D image edition [80].

Achlioptas et al. [24] proposed the first GANs architecture for point clouds. The motivation of generating point clouds lies on avoiding unnecessary transformations when the target modality is in point cloud format. This initial point cloud based GANs uses fully connected layers for the discriminator and 1D-convolutional neural networks for the generator. [81] modified the initial point cloud GANs architecture. This modified version uses graph convolutions for the generator to capture the structural information of the input and improve the generation quality. [82] uses a tree structure to rearrange the input data and make the architecture proposed in [81] computationally tractable.

3D GANs development has been focused on developing structures to improve synthesis quality and training stability. To the best of our knowledge, there are not equivalents of well known GANs frameworks compatible with 2D images in 3D GANs framework. Additionally, there are not implementation of 3D GANs in the data augmentation, anomaly detection, data privacy, and domain adaptation frameworks. This research uses proposed 3D GANs architectures with the training improvements suggested by [20] to augment 3D datasets and improve the classification performance of human actions encoded in 3D data.

4 Approach

Three dimensional data is able to better represent the reality of data and their associated problems. However, its acquisition is not as simple as with 2D images and is harder to manipulate because its large dimensions [6]. Deep learning based on convolutional neural networks is a promising methodology to process 3D data for classification problems [30] [6]. As a drawback, deep learning do not perform well with small training sets which is a common problem with 3D data as is acquisition is not straightforward.

Traditionally, data scarcity is solved using augmentation methods that slightly modify the original data [62]. Generative adversarial Networks are a promising technique to synthesise data and augment a dataset as are able to generate realistic data with not seen variations [12] [8]. Data augmentation with GANs has been done frequently with 2D data [8] [10] [65]. However, no augmentation scheme has been suggested with 3D GANs.

This research evaluates GANs as method to improve the performance of 3D based classifiers with synthesised data. To do so, the research evaluates four key aspects of the augmentation process in a classification experiment. This experiment uses a 3D deep learning classifier to map 3D frames of a human doing an action to the action that the human is doing in the frame. Then, a 3D GANs generates synthetic labelled frames that are used to create an augmented dataset to train a new deep learning classifier.

In the experiment, the first aspect to evaluate is whether the augmentation of 3D data with GANs has the ability to increase the overall performance of a 3D classifier. The second is to study if GANs are able to synthesise meaningful data for all the different classes represented in the dataset or just a specific group of labels. The third evaluates whether the number of synthetic instances that are used for the augmentation process has an impact on the classification performance. Finally, to analyse if a 3D based classifier can reach good performance in detecting actions just using 3D frames. This four analysed aspect in the experiment can be translated into research questions:

- Can Three Dimensional Generative Adversarial Networks increase the performance of deep learning models through augmentation methods? Does this increase of performance depend on the number of instances available for training?
- How many synthetic instances have to be added to the original dataset to maximise the performance of the augmentation strategy ?
- Do Generative Adversarial Networks synthesise the data from the different label with the same quality? Where quality is measured as the improvement in classification performance for the specific label
- Does volumetric data provide enough information to be used in an frame based action classifier?

To the best of our knowledge no research has evaluated the potential of 3D based GANs to augment a 3D dataset and no research has used an action recognition classifier

with a frame based approach while using volumetric data. The research opens the door to application of GANs into areas where 3D data is used such as robotics and medicine.

4.1 Dataset

This experiment uses the public available human action dataset Dynamic FAUST. The dataset was created by Bogo et al. [83] and contains 3D scans of human subjects in motion. The dataset contains information of 10 subject doing 14 different actions. The actions are represented as sequences of frames where the subjects are represented as 3D triangular meshes. Figure 16 represents an action as a sequence of frames

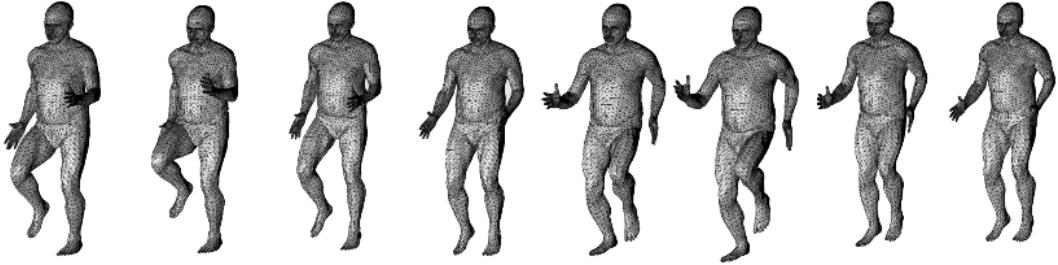


Figure 16: Dynamic FAUST: action as a sequence of frames

The dataset contains a total of 40,000 frames and the number of frames per action depends completely on the action and the subject who is doing the action. The actions represented in the dataset are: punching, running on spot, chicken wings, moving hips, moving knees, jumping jacks, shake arms, shake shoulders, shake hips, one leg loose, one leg jump, soft hop with two legs, one leg hop, and jiggling on toes. Appendix A show the label assigned to each action. Figure 17 shows a sample of the frames within the Dynamic FAUST dataset.

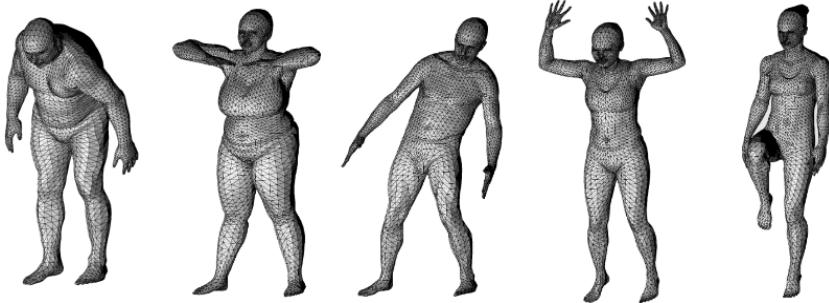


Figure 17: Dynamic FAUST frame samples

As a limitation, the dataset do not provide the equivalent 2D images of the three dimensional frames that were used to create the 3D objects. Consequently, a comparison between the classification of 3D frames and 2D frames can not be made in fair conditions.

4.2 Data Pre-Processing

The original dataset experienced two transformations; a transformation of the 3d format to represent the frames and a filtering process for the frames that are used for classification.

In first place, the original dataset format, triangular mesh, is transformed into point clouds and voxelgrids because the lack of methodologies for triangular meshes. Although there are 3D classification methodologies that use triangular meshes, these methodologies are limited to specific shapes [37] and there is not a GANs methodology compatible with the format. Whereas, other 3D classifiers that use formats such as point clouds and voxelgrids have less restrictions and are more developed than mesh based classifiers [6] [7]. Additionally, there are GANs frameworks for point clouds and voxelgrids [31] [24].

To transform triangular meshes into point clouds, the vertices of the triangles were transformed into points in a x,y,z plane and the links between vertices were deleted resulting in a point cloud. Then, point clouds are transformed into voxelgrids using the spatial occupancy method [84]. This method overlaps a grid of voxels over the point cloud space and for each voxel in the grid a binary decision is made based on whether the voxels grid is occupied by points clouds. If a voxel is occupied, the grid coordinate gets the status of occupied (1) otherwise the grid receives the status on an empty space (0). The quality and the fidelity of the voxelgrid representation increases with the dimensions of the overlapping voxel grid. However, an increase in quality increases the computation complexity as the number of voxels increases as the cube of the dimensions of the voxelgrid. This research uses voxelgrids of $32 \times 32 \times 32$ and $64 \times 64 \times 64$ size as are the standard sizes in the domain [6] [30] [31]. All the frames were transformed into point clouds and then into voxelgrids. Figure 18 illustrates the voxelisation process [85].

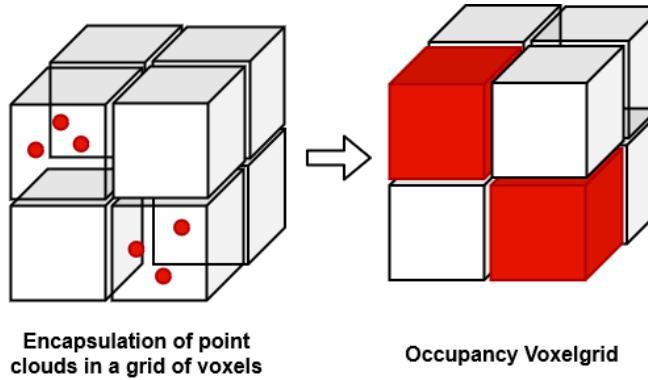


Figure 18: Voxelization process

This research uses a 3D classifier trained with frames of an action to classify a human action just with one frame. The performance of this methodology relies heavily on the quality of the information provided in the frames used for training [86]. Consequently, once the frames are transformed into point clouds and voxelgrids, the frames are filtered

to keep only informative frames.

The initial frames and the last frames of a sequence do not provide any information about the action and are removed from the whole sequence. Each action is made by a sequence of frames that represent the different situations in an action. However, in this dataset, the initial frames of an action do not contain any relevant information as the subject is in a steady state. Then, after several frames, the subject starts to perform an action. The same issue happens with the last action frames.

In addition, after removing the non representative frames, consecutive frames that present similar information are smoothed into a single frame. In the dataset actions are presented as frames of an animation animation, to produce an animation, consecutive frames have to be similar. However, these similar frames provides duplicate information. To remove the duplicate information, the frames are grouped in sequences of five consecutive frames as suggested in [87]. In each group of frames, out the five frames, one is kept in the dataset and the other four are removed. Hence, keeping differentiated frames that represent key parts of an action. After the pre-processing stage, the number of frames available in the dataset is 2634. All the action present a similar number of frames. Figure 19 represents filtered actions in the different 3D formats used in this research.

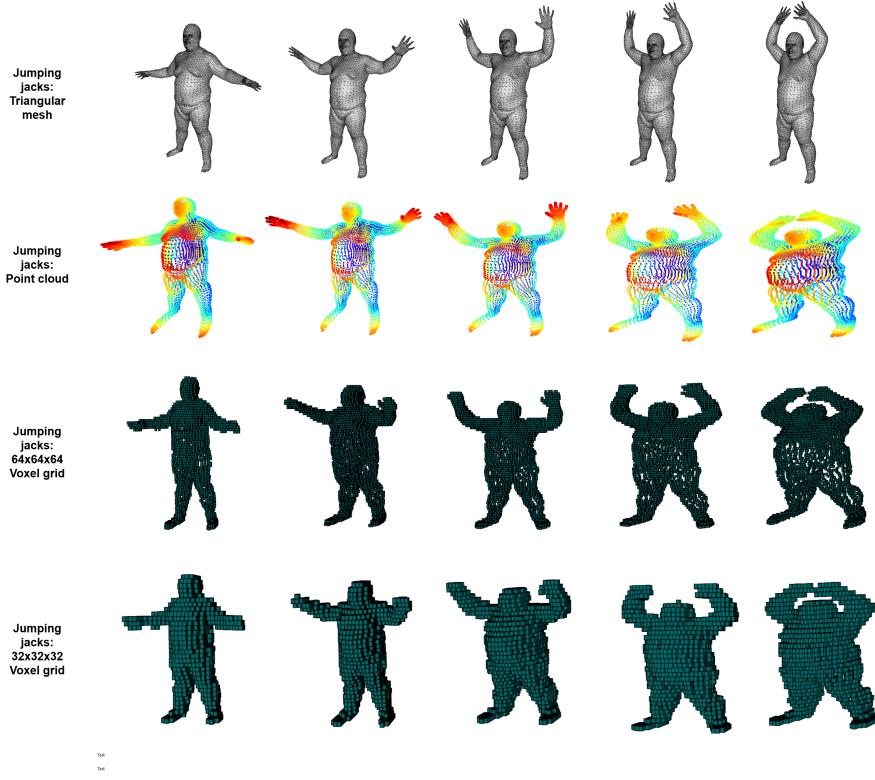


Figure 19: Frames of a sequence in different 3D formats

4.3 Dataset Split

To investigate the effect on the amount of data needed for an effective augmentation strategy. After the pre-processing stage, three different dataset with different sizes were created, namely small, medium, and big datasets. The small dataset contains 20% of the processed set ,the medium 60% and the big dataset contains all the instances available in the processed set. The medium and small set were created while keeping the same proportion of frames per action as the proportion of frames per action in the original dataset. The frames used to created the datasets were chosen randomly.

These datasets are created to evaluate the capacity of GANs to improve deep learning models in different scenarios with different data limitations. The evaluation is made by applying the proposed classifier and augmentation scheme into all three datasets. Then, comparing the impact of the data augmentation impact across datasets. In each of the three created sets, 80% of the dataset is used as a training set and 20% as a testing set.

4.4 Action Classification

The research aims to use a classifier able to handle 3D data and use it to identify frames of human actions. Consequently, evaluating the potential of classifiers that use 3D data as an input and the capacity of 3D GANs to improve classification processed with data augmentation. The proposed 3D classifier classifies one single 3D frame into an action or label. Although identifying a human action just from a single frame is possible and has showed good results, the methodology can be implemented into complex action recognition processes by classifying multiple frames of an sequence of frames and using a voting system to identify the action represented in the sequence [87]. Hence, if the classifier has a high performance in mapping a frame to an action, an action classifier based on voting multiple frames should have a high performance. Normally the number of frames used to detect an action is between 1-7 [86]. This approach has been used previously with 2d data [86]. Although the methodology has good performance, it does not use volumetric information. It is expected that the 3d information will boost the classifier as it contains valuable information. Not similar experiments have been done using 3D based classifiers.

Volumetric CNNs are an attractive methodology to do classification while considering spatial information. Volumetric CNNs tend to reach good performance compared with other classifiers. Other 3D classification methodologies such as simple classifiers, point cloud deep learning and multi-view classifiers are not in line with the project approach or perform worse than volumetric CNNs. Althought, Point clouds based classifiers avoid losing information because no data transformation is required, this methods do not explore spatial and neighbour characteristics and tend to perform worse than Volumetric grids. [7]. Simple classifiers can not handle the large dimensions of 3D representation of humans actions [6]. Finally, multi-view classifiers perform well but does not explore thoroughly the spacial characteristics of the data [34]. The major problem with volumetric CNNs is to find an structure to handle the large dimension of voxelgrid data.

Deep learning 3D CNNs structures are made of multiple layers interconnected using volumetric CNNs as a back bone layer. The most frequent layers in 3D volumetric structures are the input layer(I), fully connected layer(FC), and pooling layers (P) [6]. However, there is an unlimited number of combination of layers and hyperparameters. The 3D action classifier employed in this project follows a Voxnet architecture [6]. Voxnet has proven to reach similar performance to other volumetric CNNs structures such as Shapenet [30] but Voxnet requires a smaller number of parameters. Voxnet is a feed-forward with a layer structure $C(32,5,2) - C(32,3,1) - P(2) - FC(128) - FC(K)$ where K represents the number of classes, $C()$ a convolutional layer, P a max pooling layer and FC a fully connected layer. In C , the first parameter indicates the filter size, the second the stride and the third the padding parameters. The output of the convolutional and fully connected layers is passed through a leaky rectified non-linearity unit (ReLU) [88] with parameter 0.1. To avoid model overfitting, the output of each layer is passed through a dropout regularisation process with a dropout rate of 0.5 [89]. The last fully connected layer activation function is a softmax nonlinearity that provides a probabilistic output. Figure 20 illustrates a Voxnet architecture.

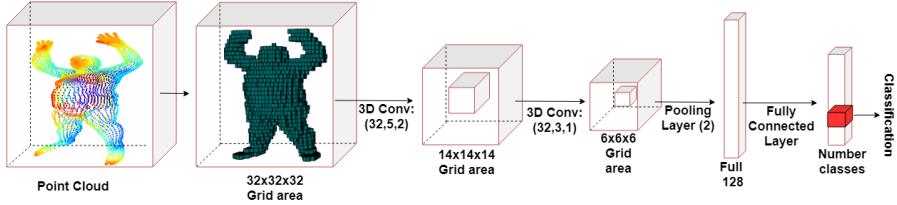


Figure 20: VoxNet Architecture

The input of a Voxnet structure is a set of voxelgrids $\mathbf{X}=\{x_1, x_2, x_3\dots x_n\}$ where x_n represent a frame of an action as grid of size $I \times J \times K$ where $I=J=K=32$. The grid values are integers in the (0-1) range where 1 represents an space occupied by a single voxel and 0 an empty space. The classifier output is a label assigned to a single instance x_n . In this experiment, the labels are the actions to classify. An instance is labelled with the class with the highest probabilistic output in the softmax non-linearity layer. The labels are the actions represented in the dataset.

The network hyperparameters of the experiment 3D classifier follow the configuration suggested in Maturana et al. [6]. The model weights are trained with Stochastic Gradient Descent with momentum rate of 0.9 and a learning rate of 0.01. The training objective function is a multinomial negative log-likelihood. The batch size is 32. The structure parameters are initialised using a zero-mean Gaussian distribution.

4.5 3D Generation & Data Augmentation

The performance of deep learning models depends on the amount and quality of the data used for training [8]. This research evaluates the ability of 3D GANs to synthesise

new data and improve the performance of 3D deep learning models. To do so, the trained GANs add synthetic 3D data from the same distribution as the original data but with unseen variations to the training set. Then, the potential of the augmented data set is evaluated by comparing the performance of a classifier trained with and without synthetic samples. The performance of the augmentation lies heavily on the configuration of the implemented GANs and the type of GANs used [10].

Depending on the format of the synthesised data, there are two types of 3D GANs; point cloud based GANs [24] and voxelgrids based GANs [31]. In this experiment, GAN generates in voxelgrid format because the classifier uses voxelgrids. Hence, avoiding loss of information when transforming the data. The structure of the generator and the discriminator is crucial for the ability of GANs to synthesise good looking images [79]. The implemented voxelgrid GANs is based on the original 3D GANS structure proposed in Wu et al.[31] and the training improvements suggested in Salimans et al. [20].

The generator is a feed forward deep learning structure made of five volumetric CNNs. The number of channels is $\{512, 256, 128, 64, 1\}$. All the volumetric CNNs have kernels of size 4 and all the layers but the first layer have a stride length of 2, the first layer has a stride of length 1. The structure includes ReLU and batch normalisation layers after every volumetric CNNs. The Generator input is a 200-size vector, this vector is retrieved from a Gaussian distribution $(0, 0.33)$ as is empirically shown that improves the model convergence and the synthesis quality [51]. The generator output is a voxelgrid matrix of $64 \times 64 \times 64$ dimension with values in the $(0-1)$ range. Although, the original 3D GANs structure suggest to use $\min \log(1 - D(G(z)))$ as generator loss function where $D(G(z))$ is the generator performance, a generator loss $\max \log D(G(z))$ is used as provides stronger gradients and avoid gradient vanishing problems [20]. Figure 21 illustrates the discriminator.

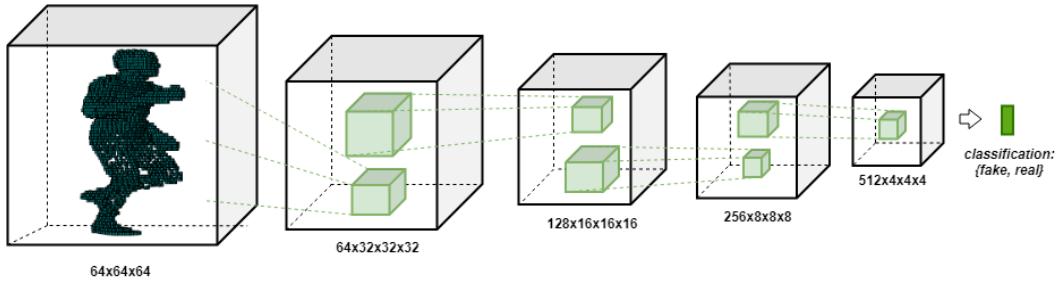


Figure 21: The discriminator in 3D-GANs

The discriminator is a feed-forward deep learning structure made of five Volumetric CNNs. The number of channels in each CNNs layer is $\{64, 128, 256, 512, 1\}$. Each volumetric convolutional layer has a kernel size of 4 and a stride length of 2, the last layer has a stride length of 1 instead of 2. In addition, there are leaky ReLU layers with parameter 0.2 and batch normalisation layers after every volumetric CNN layer. The last volumetric CNNs layer has a sigmoid activation function. The discriminator's

input is a voxelgrid matrix of $I \times J \times K$ dimensions where $I=J=K=64$. The output is in the range (0-0.9) instead of (0-1) because smooths the discriminator decision and avoids an overconfident discriminator. If the output is above 0.5 the instance is labelled as real while if the output is below 0.5 the instance is classified as fake. [20]. Figure 22 illustrates the discriminator.

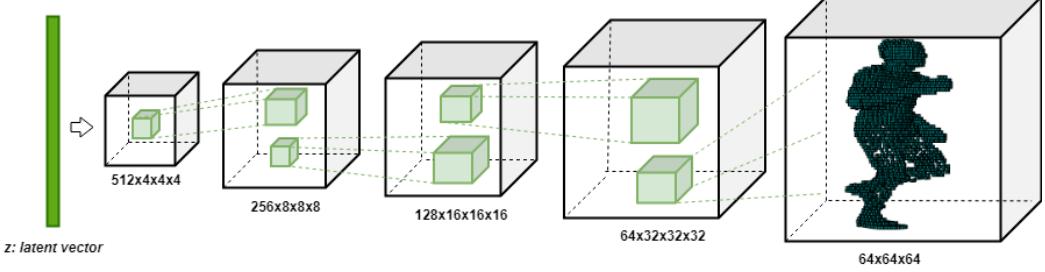


Figure 22: The generator in 3D-GANs

GANs are highly sensitive to the training configuration [58]. However, the lack of a standard method to measure synthesis quality and the long training process complicate the hyperparameters tuning process. In this project, the hyperparameters configuration is based on the original configuration with few modifications. The model parameters are trained with ADAM optimiser [21] with a $\beta = 0.5$. The discriminator batch size is 32. The model parameters are initialised using Xavier initialisation method [90]. Figure 23 represents the assembled 3D GANs structure.

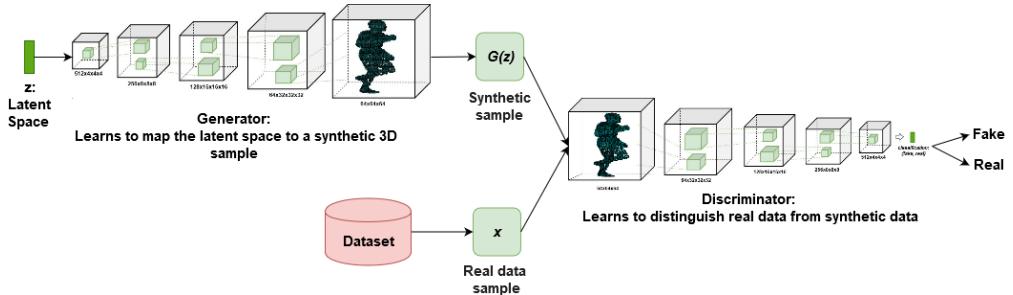


Figure 23: 3D-GAN standard structure

A common problem in 3D GANs is the discriminator learning quicker than the generator because the generation of voxelgrids is harder than distinguishing between synthesised and real voxelgrids [43]. This leads to the discriminator to differentiate instances perfectly while not issuing gradients. Without gradients, the generator cannot be updated resulting in a vanishing gradient [51]. To regulate the learning pace, this experiment GANs implements an adaptive training strategy [31] where the discriminator is updated only if the discriminator accuracy of the last batch is below 80%.

The GANs were trained with a generator learning rate of 0.0025 and a discriminator learning rate of 0.00005 as suggested in the original 3D GANs configuration. Other learning rates were analysed, if the discriminator learning rate is above 0.00005 the model tends to fall into a vanishing gradient. Whereas, if the discriminator learning rate is below 0.00005 the model has a lower synthesis quality. Figure 24 shows the evolution of the GANs loss functions and discriminator accuracy when the discriminator learning rate is above 0.00005. In this figure the discriminator learns faster than the generator because the superior learning rate. The accuracy is always above 0.5 and the discriminator loss is always close to 0 leading to a gradient vanishing problem.

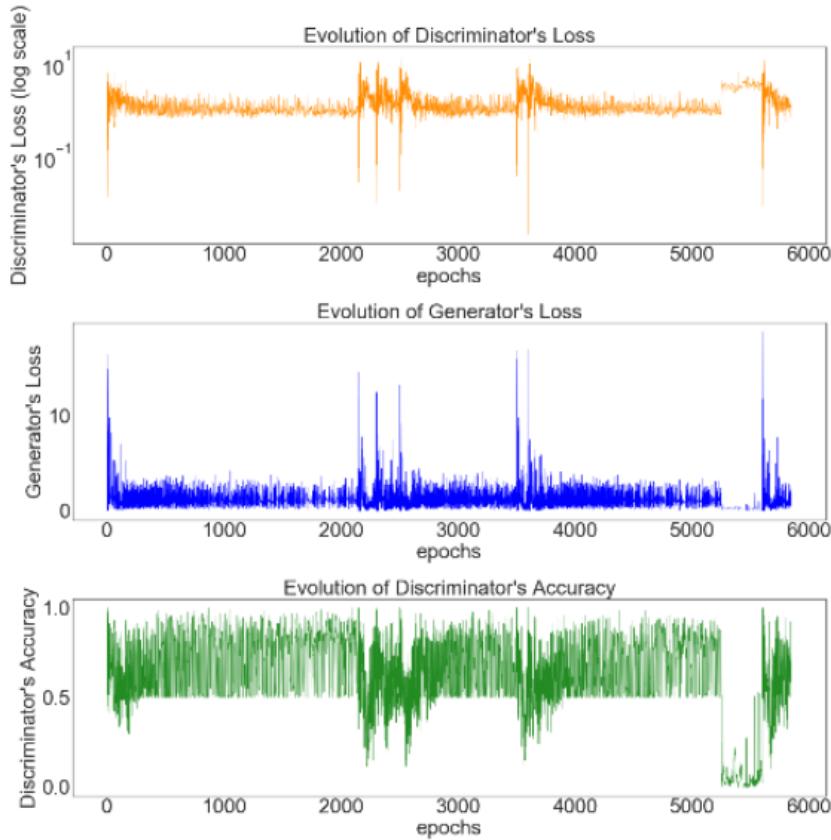


Figure 24: GANs training evolution with high discriminator learning rate

A frequent stopping criteria in GANs is to stop the training process when there is not improvement in synthesis quality [12]. During this research 3D GANs training process, there is a point where the discriminator starts to learn at a faster pace than the generator despite the measures applied to avoid it. This triggers a vanishing gradient that leads to a continuous decrease of synthesis quality. Figure 25 shows the evolution of the discriminator accuracy in two GANs training process. In the first one, the discriminator gets a continuous accuracy of 100% after the 3500 epoch. Whereas in the second training

process, the GANs enter into a vanishing gradient after the 4000 epoch.

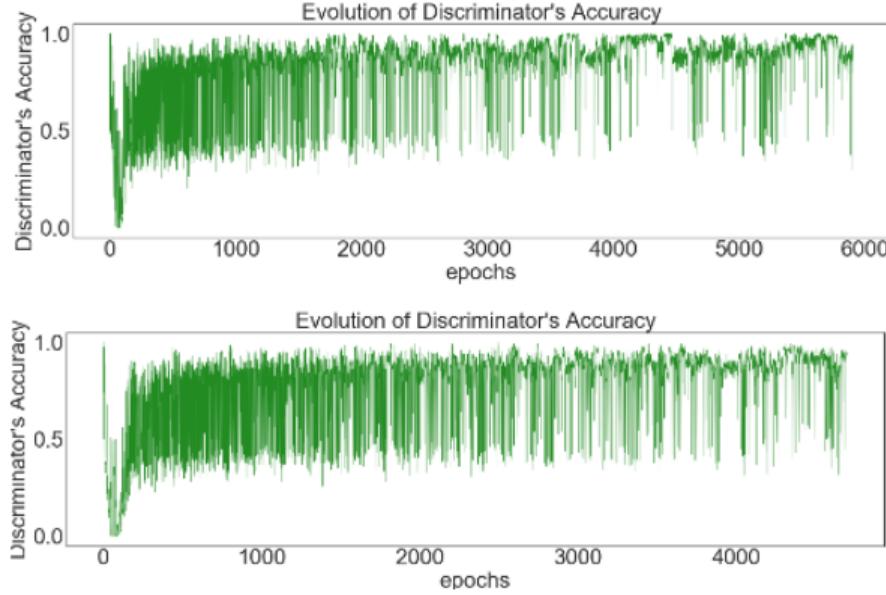


Figure 25: Vanishing Gradient in two GANs training process

The synthesis quality peaks in the epochs before the generator turning into a strict discriminative behaviour. This event happens for every label and in all the datasets in the experiment. Consequently, in this project, the stopping criteria is the generator reaching a constant accuracy of 100%. Figure 26 shows the evolution of the quality of the synthesised data before the training process reaches a vanishing gradient and after.

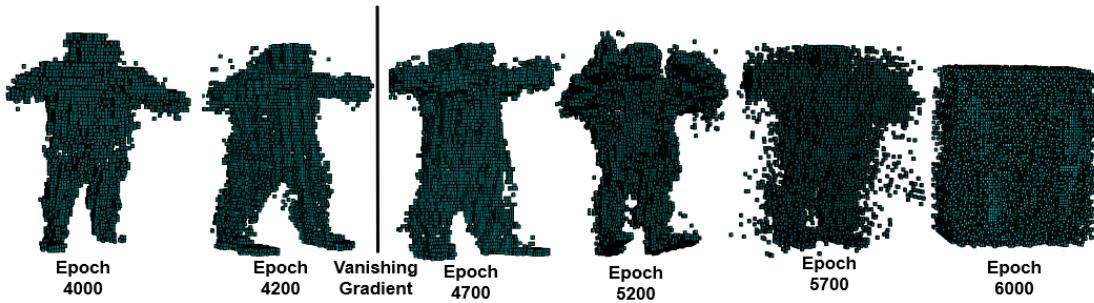


Figure 26: Synthesis quality: Vanishing Gradient

To generate labelled data, GANs are trained only using the data that belong to a class. In addition, training one GANs per class reduces the risk of the generator synthesising only few instances types to fool the discriminator [51]. As a result, multiple

individual GANs are trained for every label in each dataset. The augmentation process finishes when the synthesised labelled data is added to the original dataset as a training data. Figure 27 illustrates the data augmentation process with GANs

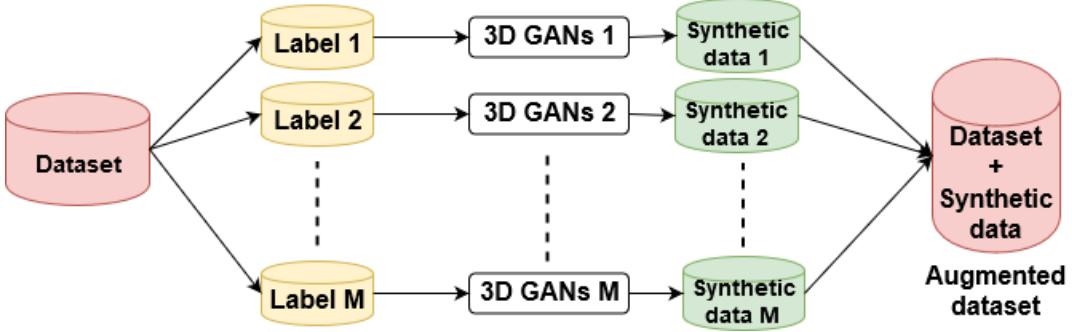


Figure 27: Data Augmentation Process

4.6 Evaluation

The proposed classification methodology is trained with a training set made of 80% of the instances of the processed set while 20% of the training set is used as a held out validation set. The classifier is evaluated with a testing set made of 20% of the processed data. The validation set is used to track whether the model is overfitting and to evaluate, during the training process, the best epoch to stop the training. The resulting model is the configuration of the model in the epoch of the training process with the highest validation accuracy. Figure 28 shows the evolution of the training and validation set.

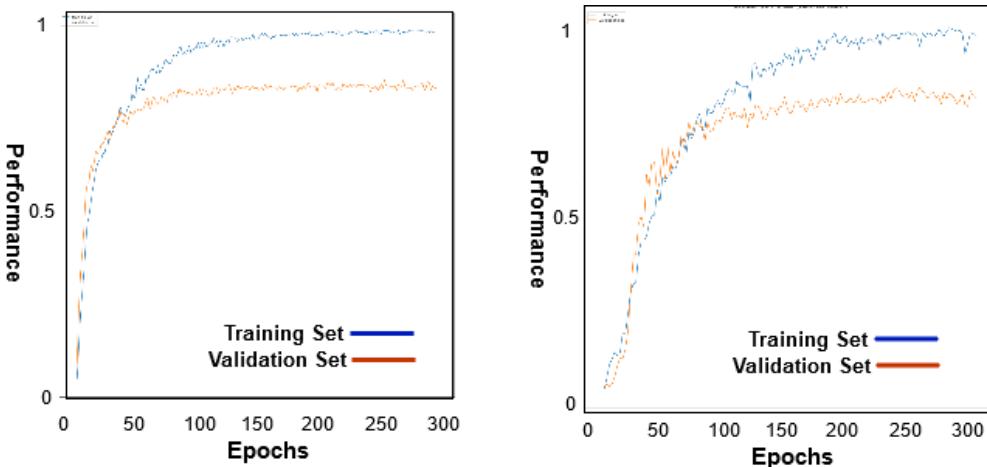


Figure 28: Example of Validation Evolution of 3D Classifier

Accuracy and confusion matrix are used to evaluate the classification. The evaluation follows a 10 fold cross validation. The accuracy reports the overall performance of the model while the confusion matrix is reported to evaluate the performance of the classifier for each individual action. The performance of the classifier is reported for the three proposed datasets, namely small, medium, and big. The comparison of the performance across datasets will show how the 3D classification of actions is affected by the size of the dataset.

To evaluate the impact of the data augmentation process, GANs are trained with the training set of each of the datasets. The output of the GANs is added to the training set to create an augmented set as showed in figure 27. Then, the 3D classifier is trained with the same approach as the non augmented data. However, the model is trained with the synthesised and training data. The number of synthetic instances to add is determined by the augmentation that returns the best performance in the classification stage. The performance of the classifier trained with and without augmented dataset are compared to evaluate the potential of an augmentation process made with GANs in different scenarios. Additionally, the confusion matrices of the augmented and non augmented dataset are compared to evaluate whether GANs can improve the action detection of all the actions or just detection of a selected number of action. Hence prove if GANs synthesise all the label with the same quality or just instances of specific actions. In the augmented sets, the 10-fold validation is ensured to not use synthetic data as testing data. Figure 29 summarises this project stages

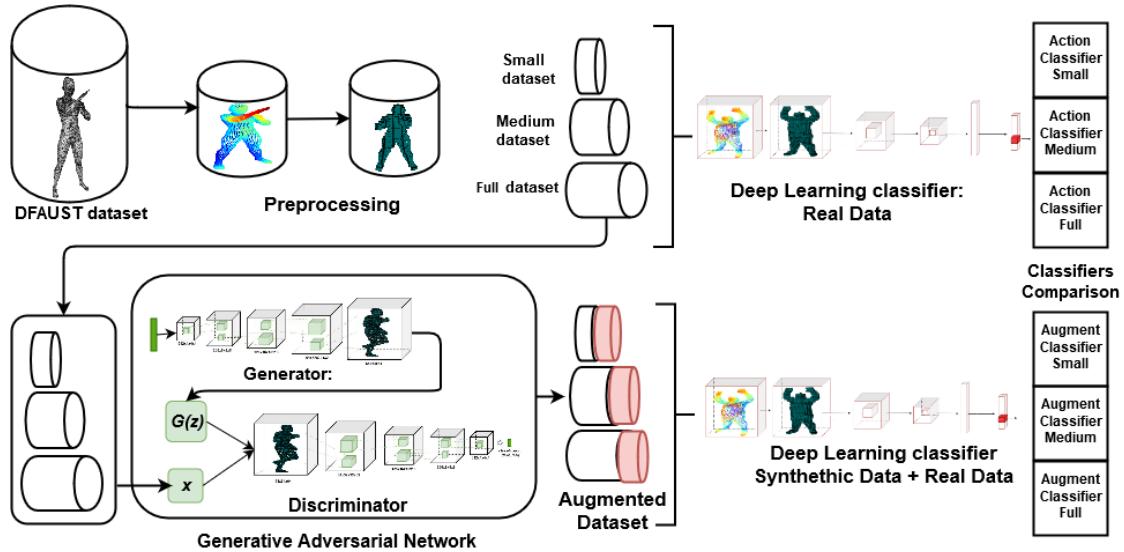


Figure 29: Project pipeline

5 Results

5.1 Qualitative Analysis

A GANs with the configuration stated in section 4.5 was trained for each label in each of the three dataset. Resulting in multiple models able to synthesised new labelled data. An initial analysis was made to evaluate the variety and quality of the generated data.

The initial visual analysis of the synthesised data reveals that, visually, there is no difference between the synthesised data from the three different dataset. Additionally, the GANs did not enter into a complete model collapse state as the trained GANs are able to generate different variation for each label. Figure 30 illustrates synthesised data samples where the object in each row belong to the same class. In each row, the first three objects are the synthetic and the last two are original objects.

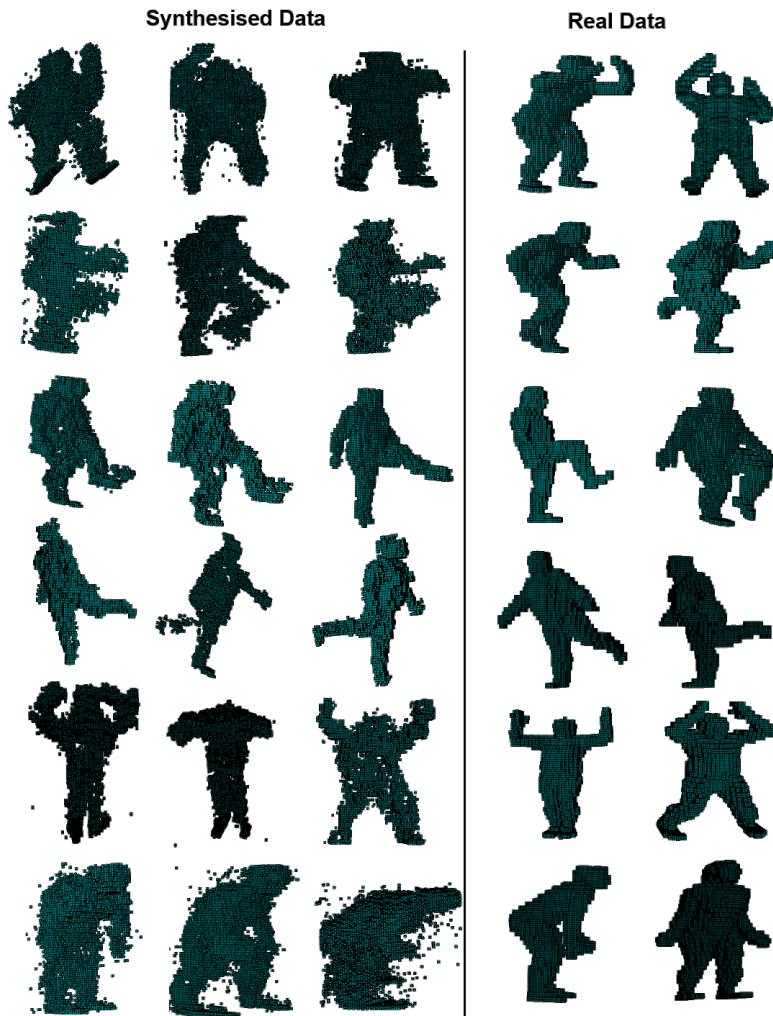


Figure 30: Real and Synthethic data sample

The synthesised objects are similar, but not identical, to the original samples. Although these imperfections, the synthesised 3D objects show that empirically the generator is able to represent the distribution of complex 3D models.

5.2 Augmentation Size

Once GANs are trained, there is a complete control over the number of instances to synthesise. Although, an unlimited number of synthetic samples can be added to the original set, the number of samples to add should be considered. Augment the dataset with excessive synthetic samples saturates the classifiers with similar information and increases the chances of training a model that does not generalise well. Contrarily, augmenting a dataset with few synthetic instances do not employ the potential of the augmentation. In this project, the augmentation size impact is evaluated by comparing the accuracy of the proposed classifiers trained with multiple augmentation schemes.

The experiment tries five different augmentation sizes for each dataset. The original datasets are augmented with synthetic data in a proportional number of the size of the original set, with 10%, 20%, 30%, 40%, and 50% as used proportions. The number of synthetic instances per each class added to the original set was considered to keep the proportion of classes as in the original dataset. Table 1 shows the accuracy results of the data augmentation in each of the three different dataset for each of the proposed augmentation sizes. The average accuracy and the standard are reported as the experiment is repeated several times with the cross validation evaluation method.

Percentage Augmented	10%	20%	30%	40%	50%
Average Accuracy in Small Dataset	0.759	0.759	0.791	0.797	0.794
Standard Deviation of Accuracy between experiments Small Set	0.015	0.026	0.021	0.01	0.019
Average Accuracy in Medium Dataset	0.834	0.829	0.858	0.861	0.858
Standard Deviation Accuracy between experiments Medium Set	0.012	0.008	0.009	0.007	0.009
Average Accuracy in Full Dataset	0.897	0.9	0.909	0.916	0.913
Standard Deviation of Accuracy between experiments in Full Set	0.002	0.005	0.007	0.005	0.008

Table 1: Accuracy of 3D classifier with multiple augmented sets

In all the datasets, the accuracy improved as the number of synthetics samples used for augmentation increased, up to an augmentation proportion of 40% of the size of the original set. Above this proportion, when more synthesised data is added, the augmentation fails to improve the accuracy. The classifier accuracy decreases because the synthetic data cannot provide more meaningful information and the classifier stars to be feed up with similar information. Proving that the number of synthetic samples added to the original set has an impact on the performance, this research uses the best augmentation policies for further comparisons. Figure 31 illustrates the increase of

performance with the different augmentation strategies in each different data sets.

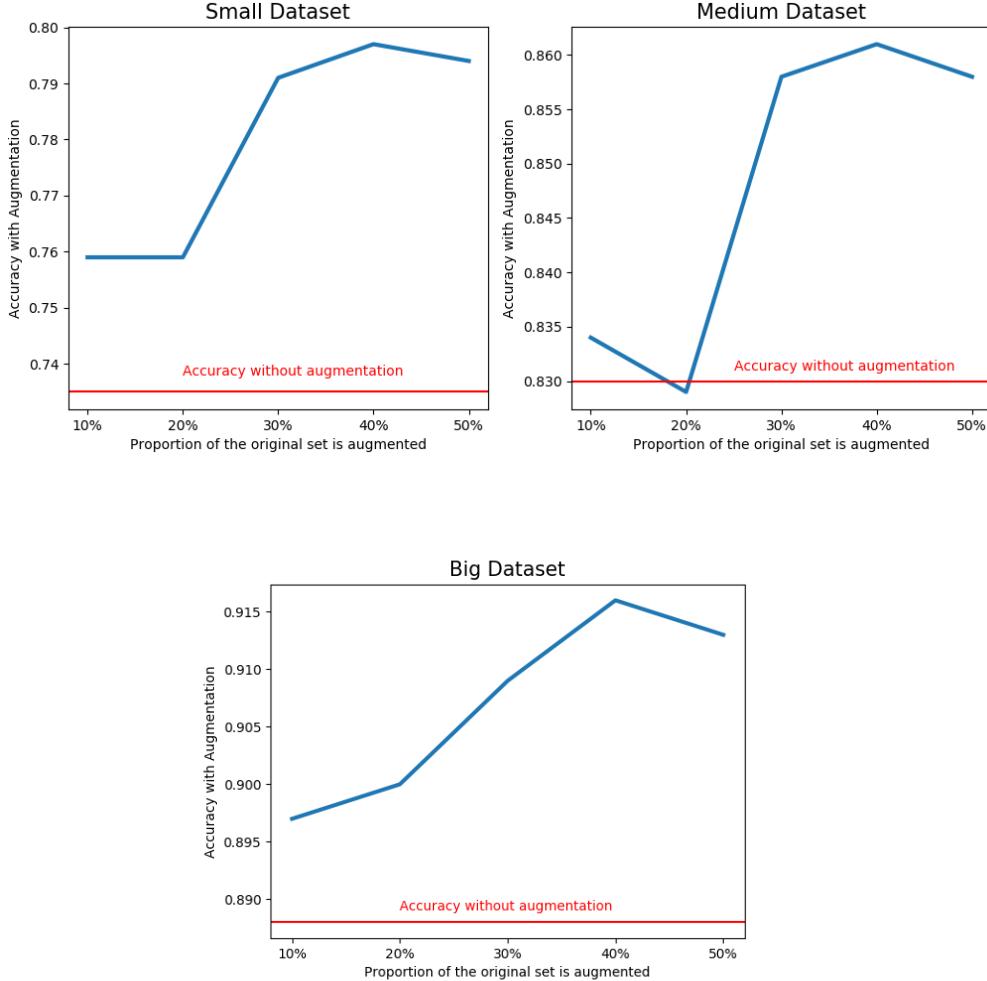


Figure 31: Accuracy fluctuation among the proposed augmentation strategies

5.3 Overall Results

The potential of the proposed augmentation strategy to improve the classification of 3D data is tested by comparing the performance of a 3D classifier trained with the original and augmented datasets. The impact of the dataset size on the performance of the augmentation methodology is tested by comparing the performance fluctuation due to the augmentation across the proposed sets. Table 2 shows the average accuracy and the standard deviation results of the multiple repetitions of the 3D classifier trained with the original and the augmented datasets in each of different dataset. The table, also, reports the fluctuation of performance between the augmented and baseline classifiers.

	Small Dataset	Medium Dataset	Full Dataset
Average Accuracy in Standard Classifier	0.735	0.83	0.888
Standard Deviation StandardClassifier	0.027	0.006	0.006
Average Accuracy in Augmented Classifier	0.797	0.861	0.916
Standard Deviation Augmented Classifier	0.01	0.007	0.005
% Increase of Accuracy with Augmentation	+8.43%	+3.73%	+3.15%

Table 2: Performance comparison between augmented classifiers and non augmented

The proposed augmentation improves the classification performance in all the datasets, increasing the classification accuracy in the big dataset by 3.15% (88.8% to 91.6%) and in the medium by 3.75% (83% to 86.1%). The proposed augmentation methodology performs particularly well in small datasets, the accuracy in the small set increased by 8.43% (73.5% to 79.7%). Figure 32 illustrates the evolution the performance across the augmented an non augmented sets.

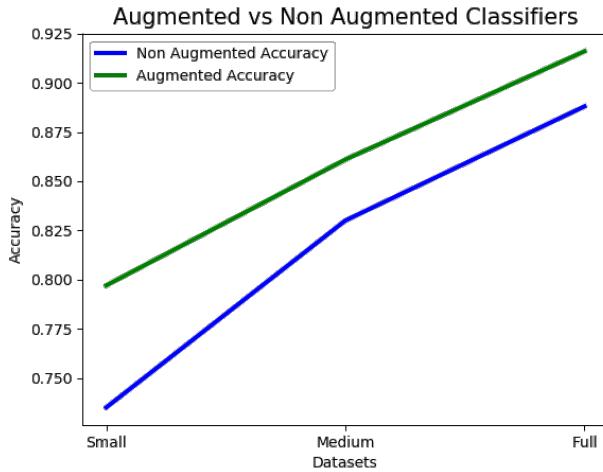


Figure 32: Augmented Classifiers vs Standard Classifiers

The results of the augmentation process confirms that, in overall, GANs can generate data from multiple distributions and use these representation to improve the performance of 3D classifiers. However, not all the distribution are equally easy to reproduce and some classes might be wrongly represented. This research evaluates whether GANs are able to represent all the distributions by comparing the classification performance of each single class between the augmented and normal classifier across the different datasets. Then, identify the wrongly represented classes to improve the model performance a posteriori.

The percentage of instances that the 3D classifier correctly classifies per class is retrieved from the normalised confusion matrices of each model. Then, the class performance is compared between the augmented and original datasets in all three dataset variants. Appendix B shows the normalised confusion matrices for each model. Tables 3, 4, and 5 show the percentage of instances properly identified in each class for the augmented and non augmented classifiers. The tables also provides information about

the fluctuation in single class accuracy between augmented and non augmented cases.

Label	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Label Accuracy Small Classifier	0.57	1.00	0.57	0.67	0.85	0.75	0.88	0.63	0.78	0.70	0.50	0.67	0.88	0.78
Label Accuracy Small Augmented Classifier	0.57	1.00	0.71	0.67	0.92	0.67	0.88	0.88	0.56	0.70	0.71	0.73	1.00	0.78
Small dataset Increase label Accuracy	0.00	0.00	0.25	0.00	0.09	-0.11	0.00	0.40	-0.29	0.00	0.43	0.09	0.14	0.00

Table 3: Small Dataset Label Accuracy comparison

Label	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Label Accuracy Medium Classifier	0.73	0.97	1.00	0.97	0.88	0.56	0.77	0.83	0.86	0.71	0.61	0.74	0.92	0.89
Label Accuracy Medium Augmented Classifier	0.82	0.97	1.00	0.97	1.00	0.60	0.82	1.00	0.77	0.75	0.75	0.85	0.96	0.93
Medium dataset Increase label Accuracy	0.13	0.00	0.00	0.00	0.14	0.07	0.06	0.21	-0.11	0.05	0.23	0.15	0.04	0.04

Table 4: Medium Dataset Label Accuracy comparison

Label	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Label Accuracy Full Classifier	0.86	0.86	0.97	0.86	0.97	0.86	0.95	0.89	0.93	0.85	0.71	0.87	0.98	0.87
Label Accuracy Full Augmented Classifier	0.97	0.97	0.94	0.91	0.94	0.85	0.95	0.97	0.93	0.96	0.85	0.91	0.98	0.85
Full dataset Increase Accuracy	0.13	0.12	-0.03	0.06	-0.03	0.00	0.00	0.09	0.00	0.12	0.21	0.05	0.00	-0.02

Table 5: Full Dataset Label Accuracy comparison

In overall, the classification performance of the individual labels increases with the augmentation process showing the ability of GANs to model the distributions of multiple situations and synthesise non-seen data from those distributions. There are classes such as 3 (moving hips), 6 (shake arm), and 13 (jiggling on toes) whose classification performance remains invariant after the augmentation. Contrarily, the classification performance of class 8 (shake hips) decreases as a result of the augmentation. This shows that, certainly, there are distributions that are harder to learn and either GANs cannot model those distributions or require a different configuration of parameters to learn. One simple solution is to avoid using GANs to synthesise labelled data for those classes that GANs cannot learn their distribution. However, knowing a priori that GANs cannot generate a distribution properly is a complicated task as there is not a good measure to evaluate the fidelity of the GANs learned distributions [20].

Overall, the results confirm that the proposed classifier detects actions using 3D objects even when the number of data available is limited. The presented augmentation strategy improves the classification performance of classifiers trained on datasets of all sizes, particularly, small dataset. The efficacy of the augmentation strategy depends on the number of synthetic samples added to the original set. Finally, GANs are not always able to represent all the distributions within a dataset which reduces the performance of the augmentation strategy if wrong representations are included in the training set.

6 Conclusion

This research proposes a method to synthesise 3D human representations using 3D generative adversarial networks to improve the performance of deep learning models for the classification of 3D images. The research outlines that structural modifications of the original 3D GANs structure improves the generation quality of complex 3D data distributions. The proposed 3D GANs learns the distributions within a dataset to introduce novel labelled objects into the training set of a deep learning classifier. This results in improvements in classification performance in all kinds of datasets, particularly, in low-data settings. However beyond that, this research shows the limitations of GANs to produce a variety of non-seen information. Consequently, the number of synthetic instances used for augmentation should be considered. Furthermore, this research identifies that GANs can learn incorrectly some distributions within a dataset which leads to a performance decrease of the augmentation. Finally, the good results obtained by the proposed 3D classifiers and the ability of 3D based GANs to learn 3D data distributions for its generation confirms the suitability of 3D data to represent information.

6.1 Further Research

Future work should be focused on adapting 2D based GANs structures into 3D based GANs. 2D based GANs structures able to perform domain transfer such as cycle GANs or pix2pix GANs have shown good results on augmenting small datasets. Another research path lies on comparing the performance of GANs based on the generation of voxels and GANs based on the generation of point clouds for the augmentation of 3D datasets. Additionally, the development of methods to augment datasets only with meaningful synthesised samples or a method to evaluate GANs learned distributions might have a positive impact on the augmentation process. Finally, the proposed strategy needs to be evaluated in other domains where the implementation of 3D data will lead to improvements. The suggested methodology is potentially suitable for domains such as human robot interaction to improve agents perception of the real world or in medicine to boost the detection of diseases with 3D data provided by scans, body sensors, and wearables.

References

- [1] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, *et al.*, “Shapenet: An information-rich 3d model repository,” *arXiv preprint arXiv:1512.03012*, 2015.
- [2] M.-D. Yang, C.-F. Chao, K.-S. Huang, L.-Y. Lu, and Y.-P. Chen, “Image-based 3d scene reconstruction and exploration in augmented reality,” *Automation in Construction*, vol. 33, pp. 48–60, 2013.
- [3] F. Argelaguet and C. Andujar, “A survey of 3d object selection techniques for virtual environments,” *Computers & Graphics*, vol. 37, no. 3, pp. 121–136, 2013.
- [4] A. Payan and G. Montana, “Predicting alzheimer’s disease: a neuroimaging study with 3d convolutional neural networks,” *arXiv preprint arXiv:1502.02506*, 2015.
- [5] L.-Y. Gui, K. Zhang, Y.-X. Wang, X. Liang, J. M. Moura, and M. Veloso, “Teaching robots to predict human motion,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 562–567, IEEE, 2018.
- [6] D. Maturana and S. Scherer, “Voxnet: A 3d convolutional neural network for real-time object recognition,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 922–928, IEEE, 2015.
- [7] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “Pointnet: Deep learning on point sets for 3d classification and segmentation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 652–660, 2017.
- [8] A. Antoniou, A. Storkey, and H. Edwards, “Data augmentation generative adversarial networks,” *arXiv preprint arXiv:1711.04340*, 2017.
- [9] M. Frid-Adar, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan, “Synthetic data augmentation using gan for improved liver lesion classification,” in *2018 IEEE 15th international symposium on biomedical imaging (ISBI 2018)*, pp. 289–293, IEEE, 2018.
- [10] H.-C. Shin, N. A. Tenenholtz, J. K. Rogers, C. G. Schwarz, M. L. Senjem, J. L. Gunter, K. P. Andriole, and M. Michalski, “Medical image synthesis for data augmentation and anonymization using generative adversarial networks,” in *International Workshop on Simulation and Synthesis in Medical Imaging*, pp. 1–11, Springer, 2018.
- [11] H. Huang, E. Kalogerakis, and B. Marlin, “Analysis and synthesis of 3d shape families via deep-learned generative models of surfaces,” in *Computer Graphics Forum*, vol. 34, pp. 25–38, Wiley Online Library, 2015.

- [12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, pp. 2672–2680, 2014.
- [13] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro, “High-resolution image synthesis and semantic manipulation with conditional gans,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8798–8807, 2018.
- [14] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired image-to-image translation using cycle-consistent adversarial networks,” in *Proceedings of the IEEE international conference on computer vision*, pp. 2223–2232, 2017.
- [15] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar, “Efficient gan-based anomaly detection,” *arXiv preprint arXiv:1802.06222*, 2018.
- [16] J. Donahue, P. Krähenbühl, and T. Darrell, “Adversarial feature learning,” *arXiv preprint arXiv:1605.09782*, 2016.
- [17] S. Pascual, A. Bonafonte, and J. Serra, “Segan: Speech enhancement generative adversarial network,” *arXiv preprint arXiv:1703.09452*, 2017.
- [18] W. Fedus, I. Goodfellow, and A. M. Dai, “Maskgan: better text generation via filling in the_,” *arXiv preprint arXiv:1801.07736*, 2018.
- [19] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, “Netgan: Generating graphs via random walks,” *arXiv preprint arXiv:1803.00816*, 2018.
- [20] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training gans,” in *Advances in neural information processing systems*, pp. 2234–2242, 2016.
- [21] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [22] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” *arXiv preprint arXiv:1411.1784*, 2014.
- [23] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville, “Adversarially learned inference,” *arXiv preprint arXiv:1606.00704*, 2016.
- [24] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, “Learning representations and generative models for 3d point clouds,” *arXiv preprint arXiv:1707.02392*, 2017.
- [25] C.-H. Lin, C. Kong, and S. Lucey, “Learning efficient point cloud generation for dense 3d object reconstruction,” in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

- [26] G. Chételat, B. Landeau, F. Eustache, F. Mézenge, F. Viader, V. de La Sayette, B. Desgranges, and J.-C. Baron, “Using voxel-based morphometry to map the structural changes associated with rapid conversion in mci: a longitudinal mri study,” *Neuroimage*, vol. 27, no. 4, pp. 934–946, 2005.
- [27] D. Maturana and S. Scherer, “3d convolutional neural networks for landing zone detection from lidar,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3471–3478, IEEE, 2015.
- [28] M. De Deuge, A. Quadros, C. Hung, and B. Douillard, “Unsupervised feature learning for classification of outdoor 3d scans,” in *Australasian Conference on Robotics and Automation*, vol. 2, p. 1, 2013.
- [29] Y. Fang, J. Xie, G. Dai, M. Wang, F. Zhu, T. Xu, and E. Wong, “3d deep shape descriptor,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2319–2328, 2015.
- [30] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, “3d shapenets: A deep representation for volumetric shapes,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1912–1920, 2015.
- [31] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum, “Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling,” in *Advances in neural information processing systems*, pp. 82–90, 2016.
- [32] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, “Generative and discriminative voxel modeling with convolutional neural networks,” *arXiv preprint arXiv:1608.04236*, 2016.
- [33] S. Ji, W. Xu, M. Yang, and K. Yu, “3d convolutional neural networks for human action recognition,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 221–231, 2012.
- [34] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. J. Guibas, “Volumetric and multi-view cnns for object classification on 3d data,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5648–5656, 2016.
- [35] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, “Multi-view convolutional neural networks for 3d shape recognition,” in *Proceedings of the IEEE international conference on computer vision*, pp. 945–953, 2015.
- [36] V. Hegde and R. Zadeh, “Fusionnet: 3d object classification using multiple data representations,” *arXiv preprint arXiv:1607.05695*, 2016.
- [37] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, “Spectral networks and locally connected networks on graphs,” *arXiv preprint arXiv:1312.6203*, 2013.

- [38] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst, “Geodesic convolutional neural networks on riemannian manifolds,” in *Proceedings of the IEEE international conference on computer vision workshops*, pp. 37–45, 2015.
- [39] R. Klokov and V. Lempitsky, “Escape from cells: Deep kd-networks for the recognition of 3d point cloud models,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 863–872, 2017.
- [40] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “Pointnet++: Deep hierarchical feature learning on point sets in a metric space,” in *Advances in neural information processing systems*, pp. 5099–5108, 2017.
- [41] I. Goodfellow, “Nips 2016 tutorial: Generative adversarial networks,” *arXiv preprint arXiv:1701.00160*, 2016.
- [42] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, “Generative adversarial networks: An overview,” *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018.
- [43] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *arXiv preprint arXiv:1511.06434*, 2015.
- [44] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, “Adversarial autoencoders,” *arXiv preprint arXiv:1511.05644*, 2015.
- [45] C. Finn, P. Christiano, P. Abbeel, and S. Levine, “A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models,” *arXiv preprint arXiv:1611.03852*, 2016.
- [46] C. Finn and S. Levine, “Deep visual foresight for planning robot motion,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2786–2793, IEEE, 2017.
- [47] A. Odena, “Semi-supervised learning with generative adversarial networks,” *arXiv preprint arXiv:1606.01583*, 2016.
- [48] J. Ho and S. Ermon, “Generative adversarial imitation learning,” in *Advances in neural information processing systems*, pp. 4565–4573, 2016.
- [49] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, “Unsupervised anomaly detection with generative adversarial networks to guide marker discovery,” in *International Conference on Information Processing in Medical Imaging*, pp. 146–157, Springer, 2017.
- [50] D. Yoo, N. Kim, S. Park, A. S. Paek, and I. S. Kweon, “Pixel-level domain transfer,” in *European Conference on Computer Vision*, pp. 517–532, Springer, 2016.

- [51] M. Arjovsky and L. Bottou, “Towards Principled Methods for Training Generative Adversarial Networks,” *arXiv e-prints*, p. arXiv:1701.04862, Jan 2017.
- [52] S. Arora, R. Ge, Y. Liang, T. Ma, and Y. Zhang, “Generalization and equilibrium in generative adversarial nets (gans),” in *Proceedings of the 34th International Conference on Machine Learning- Volume 70*, pp. 224–232, JMLR. org, 2017.
- [53] I. O. Tolstikhin, S. Gelly, O. Bousquet, C.-J. Simon-Gabriel, and B. Schölkopf, “Adagan: Boosting generative models,” in *Advances in Neural Information Processing Systems*, pp. 5424–5433, 2017.
- [54] C. K. Sønderby, J. Caballero, L. Theis, W. Shi, and F. Huszár, “Amortised map inference for image super-resolution,” *arXiv preprint arXiv:1610.04490*, 2016.
- [55] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein gan,” *arXiv preprint arXiv:1701.07875*, 2017.
- [56] S. Nowozin, B. Cseke, and R. Tomioka, “f-gan: Training generative neural samplers using variational divergence minimization,” in *Advances in neural information processing systems*, pp. 271–279, 2016.
- [57] Q. Chen and V. Koltun, “Photographic image synthesis with cascaded refinement networks,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1511–1520, 2017.
- [58] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet, “Are gans created equal? a large-scale study,” in *Advances in neural information processing systems*, pp. 700–709, 2018.
- [59] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.
- [60] S. Ioffe, “Batch renormalization: Towards reducing minibatch dependence in batch-normalized models,” in *Advances in neural information processing systems*, pp. 1945–1953, 2017.
- [61] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *arXiv preprint arXiv:1207.0580*, 2012.
- [62] Z. Hussain, F. Gimenez, D. Yi, and D. Rubin, “Differential data augmentation techniques for medical imaging classification tasks,” in *AMIA Annual Symposium Proceedings*, vol. 2017, p. 979, American Medical Informatics Association, 2017.
- [63] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

- [64] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.
- [65] A. Gupta, S. Venkatesh, S. Chopra, and C. Ledig, “Generative image translation for data augmentation of bone lesion pathology,” *arXiv preprint arXiv:1902.02248*, 2019.
- [66] C. Han, K. Murao, T. Noguchi, Y. Kawata, F. Uchiyama, L. Rundo, H. Nakayama, and S. Satoh, “Learning more with less: conditional pggan-based data augmentation for brain metastases detection using highly-rough annotation on mr images,” *arXiv preprint arXiv:1902.09856*, 2019.
- [67] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.
- [68] J. Dai, Y. Li, K. He, and J. Sun, “R-fcn: Object detection via region-based fully convolutional networks,” in *Advances in neural information processing systems*, pp. 379–387, 2016.
- [69] J. W. Tangelder and R. C. Veltkamp, “A survey of content based 3d shape retrieval methods,” in *Proceedings Shape Modeling Applications, 2004.*, pp. 145–156, IEEE, 2004.
- [70] C. Li, M. Zeeshan Zia, Q.-H. Tran, X. Yu, G. D. Hager, and M. Chandraker, “Deep supervision with shape concepts for occlusion-aware 3d object parsing,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5465–5474, 2017.
- [71] H. Sundar, D. Silver, N. Gagvani, and S. Dickinson, “Skeleton based shape matching and retrieval,” in *2003 Shape Modeling International.*, pp. 130–139, IEEE, 2003.
- [72] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese, “3d-r2n2: A unified approach for single and multi-view 3d object reconstruction,” in *European conference on computer vision*, pp. 628–644, Springer, 2016.
- [73] R. Girdhar, D. F. Fouhey, M. Rodriguez, and A. Gupta, “Learning a predictable and generative vector representation for objects,” in *European Conference on Computer Vision*, pp. 484–499, Springer, 2016.
- [74] Y. Zhao, T. Birdal, H. Deng, and F. Tombari, “3d point capsule networks,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
- [75] X. Yan, J. Yang, E. Yumer, Y. Guo, and H. Lee, “Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision,” in *Advances in Neural Information Processing Systems*, pp. 1696–1704, 2016.

- [76] D. J. Rezende, S. A. Eslami, S. Mohamed, P. Battaglia, M. Jaderberg, and N. Heess, “Unsupervised learning of 3d structure from images,” in *Advances in Neural Information Processing Systems*, pp. 4996–5004, 2016.
- [77] Y. Xiang, W. Choi, Y. Lin, and S. Savarese, “Data-driven 3d voxel patterns for object category recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1903–1911, 2015.
- [78] A. Sharma, O. Grau, and M. Fritz, “Vconv-dae: Deep volumetric shape learning without object labels,” in *European Conference on Computer Vision*, pp. 236–250, Springer, 2016.
- [79] E. Smith and D. Meger, “Improved adversarial systems for 3d object generation and reconstruction,” *arXiv preprint arXiv:1707.09557*, 2017.
- [80] J. Liu, F. Yu, and T. Funkhouser, “Interactive 3d modeling with a generative adversarial network,” in *2017 International Conference on 3D Vision (3DV)*, pp. 126–134, IEEE, 2017.
- [81] D. Valsesia, G. Fracastoro, and E. Magli, “Learning localized generative models for 3d point clouds via graph convolution,” 2018.
- [82] D. W. Shu, S. W. Park, and J. Kwon, “3d point cloud generative adversarial network based on tree structured graph convolutions,” *arXiv preprint arXiv:1905.06292*, 2019.
- [83] F. Bogo, J. Romero, G. Pons-Moll, and M. J. Black, “Dynamic faust: Registering human bodies in motion,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6233–6242, 2017.
- [84] M. W. Jones and R. Satherley, “Shape representation using space filled sub-voxel distance fields,” in *Proceedings international conference on shape modeling and applications*, pp. 316–325, IEEE, 2001.
- [85] S. Fang and H. Chen, “Hardware accelerated voxelization,” *Computers & Graphics*, vol. 24, no. 3, pp. 433–442, 2000.
- [86] K. Schindler and L. J. Van Gool, “Action snippets: How many frames does human action recognition require?,” in *CVPR*, vol. 1, pp. 3–2, 2008.
- [87] A. Fathi and G. Mori, “Action recognition by learning mid-level motion features,” in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, IEEE, 2008.
- [88] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

- [89] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [90] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

A Table Number corresponding with the classified actions

Label	Action
0	punching
1	running on spot
2	chicken wings
3	moving hips
4	moving knees
5	jumping jacks
6	shake arms
7	shake shoulders
8	shake hips
9	one leg loose
10	one leg jump
11	soft hop with two legs
12	one leg hop
13	jiggling on toes

Table 6: Label assigned to each action

B Confusion Matrices

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.5714	0	0	0.2857	0	0	0	0.1429	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
2	0.1429	0	0.5714	0	0	0	0	0	0.1429	0	0	0.1429	0	0
3	0	0	0	0.6667	0.1667	0	0	0	0	0	0	0.0833	0.0833	0
4	0	0	0	0	0.8462	0	0	0	0	0.0769	0.0769	0	0	0
5	0	0	0	0	0	0.75	0	0	0	0	0	0.125	0	0.125
6	0	0	0.125	0	0	0	0.875	0	0	0	0	0	0	0
7	0	0.125	0	0.125	0	0	0	0.625	0	0	0	0.125	0	0
8	0	0	0	0	0	0	0.1111	0	0.7778	0	0	0	0.1111	0
9	0	0	0	0	0	0	0	0	0	0.7	0.2	0	0.1	0
10	0	0.1667	0	0	0.0833	0.0833	0	0	0	0.0833	0.5	0	0.0833	0
11	0	0	0	0	0	0	0	0	0	0	0.6667	0.3333	0	
12	0	0	0	0.125	0	0	0	0	0	0	0	0	0.875	0
13	0	0	0	0.1111	0	0.1111	0	0	0	0	0	0	0	0.7778

Figure 33: Normalised Confusion Matrix of the classifier trained with the small dataset

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.5714	0	0	0.4286	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0.7143	0	0	0	0	0	0.1429	0	0	0.1429	0	0
3	0	0	0	0.6667	0	0	0	0.0833	0	0	0	0.0833	0.1667	0
4	0	0	0	0	0.9231	0	0	0	0	0.0769	0	0	0	0
5	0	0.1111	0	0	0	0.6667	0	0	0	0	0	0.1111	0	0.1111
6	0.125	0	0	0	0	0	0.875	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0.875	0	0	0	0.125	0	0
8	0.1111	0	0	0.1111	0	0	0.1111	0	0.5556	0	0	0	0.1111	0
9	0	0	0	0	0	0	0	0	0	0.7	0.2	0	0.1	0
10	0	0.0714	0	0.0714	0.0714	0	0	0	0	0.0714	0.7143	0	0	0
11	0	0	0	0	0	0	0	0.0909	0	0	0.0909	0.7273	0.0909	0
12	0	0	0	0	0	0	0	0	0	0	0	1	0	0
13	0	0	0	0.1111	0	0.1111	0	0	0	0	0	0	0	0.7778

Figure 34: Normalised Confusion Matrix of the classifier trained with the augmented small dataset

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.7273	0.0455	0	0	0	0	0.0455	0	0.0909	0	0	0.0909	0	0
1	0	0.9714	0	0	0	0	0	0	0	0	0.0286	0	0	0
2	0	0	1	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0.9714	0	0	0	0.0286	0	0	0	0	0	0
4	0	0	0	0	0.875	0.025	0	0	0	0.075	0.025	0	0	0
5	0	0	0	0	0.08	0.56	0	0	0	0.04	0.16	0.04	0	0.12
6	0.0455	0.0455	0	0	0	0	0.7727	0	0.0455	0	0	0	0.0455	0.0455
7	0.0435	0	0	0.0435	0	0	0	0.8261	0	0	0	0.0435	0.0435	0
8	0.0357	0	0	0.0357	0	0	0.0357	0	0.8571	0	0	0	0	0.0357
9	0	0	0	0	0.0714	0	0	0	0	0.7143	0.2143	0	0	0
10	0	0.0278	0	0	0.1389	0	0	0	0.0278	0.1389	0.6111	0.0556	0	0
11	0	0	0	0	0	0	0	0	0	0	0.037	0.7407	0.2222	0
12	0	0	0	0.04	0	0	0	0	0	0	0	0.04	0.92	0
13	0	0	0	0	0	0	0	0	0.0357	0	0	0.0714	0.8929	0

Figure 35: Normalised Confusion Matrix of the classifier trained with the medium dataset

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.8182	0	0	0	0	0	0.0455	0.0455	0.0455	0	0	0.0455	0	0
1	0.0286	0.9714	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0.9714	0	0	0	0.0286	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0.04	0.6	0	0	0.04	0.04	0.16	0	0	0.12
6	0.0455	0	0	0	0.0455	0	0.8182	0.0455	0	0	0	0	0	0.0455
7	0	0	0	0	0	0	0	1	0	0	0	0	0	0
8	0	0	0	0.0667	0	0.0667	0.1	0	0.7667	0	0	0	0	0
9	0	0	0	0	0.1429	0	0	0	0	0.75	0.1071	0	0	0
10	0	0.0556	0	0	0.0556	0	0	0	0	0.0833	0.75	0.0556	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0.8519	0.1481	0
12	0	0	0	0	0	0	0	0	0	0	0	0.04	0.96	0
13	0	0	0	0	0	0	0	0	0	0	0	0.0714	0.9286	0

Figure 36: Normalised Confusion Matrix of the classifier trained with the augmented medium dataset

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.8611	0	0	0.0278	0	0	0.0278	0.0278	0.0278	0	0	0	0.0278	0
1	0.0172	0.8621	0	0	0.0345	0	0.0172	0	0	0.0172	0.0517	0	0	0
2	0	0	0.9714	0	0	0	0	0	0.0286	0	0	0	0	0
3	0	0.0172	0	0.8621	0.0517	0	0	0	0	0	0	0.0345	0.0345	0
4	0	0.0149	0	0.0149	0.9701	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0.0476	0.8571	0	0	0.0476	0	0.0238	0	0	0.0238
6	0	0	0.0263	0	0	0	0.9474	0	0	0	0	0	0	0.0263
7	0.0263	0	0	0.0263	0	0	0	0.8947	0	0	0	0	0.0526	0
8	0.0217	0	0	0	0	0	0.0217	0.0217	0.9348	0	0	0	0	0
9	0	0	0	0	0.0851	0	0	0	0	0.8511	0.0426	0.0213	0	0
10	0	0.0517	0	0	0.069	0	0	0	0	0.1207	0.7069	0.0517	0	0
11	0.0222	0	0	0.0222	0	0	0	0.0222	0	0	0.0222	0.8667	0.0444	0
12	0	0	0	0.0238	0	0	0	0	0	0	0	0	0.9762	0
13	0	0	0	0.0213	0.0213	0.0213	0.0213	0	0.0426	0	0	0	0	0.8723

Figure 37: Normalised Confusion Matrix of the classifier trained with the big dataset

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.971	0.000	0.029	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
1	0.000	0.966	0.000	0.000	0.034	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	0.000	0.000	0.943	0.000	0.000	0.000	0.000	0.057	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.017	0.000	0.914	0.017	0.000	0.000	0.000	0.000	0.000	0.000	0.052	0.000	0.000
4	0.000	0.000	0.000	0.000	0.940	0.000	0.000	0.015	0.000	0.015	0.030	0.000	0.000	0.000
5	0.000	0.024	0.000	0.000	0.000	0.854	0.000	0.000	0.073	0.000	0.024	0.000	0.000	0.024
6	0.000	0.000	0.026	0.000	0.000	0.000	0.947	0.000	0.000	0.000	0.000	0.000	0.000	0.026
7	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.974	0.000	0.000	0.000	0.000	0.026	0.000
8	0.043	0.000	0.000	0.000	0.000	0.000	0.022	0.000	0.935	0.000	0.000	0.000	0.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.956	0.044	0.000	0.000	0.000
10	0.000	0.016	0.000	0.016	0.016	0.000	0.000	0.000	0.000	0.066	0.852	0.016	0.016	0.000
11	0.022	0.000	0.000	0.000	0.000	0.000	0.000	0.022	0.000	0.000	0.022	0.911	0.022	0.000
12	0.000	0.000	0.000	0.024	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.976	0.000	
13	0.000	0.000	0.000	0.021	0.000	0.043	0.021	0.000	0.043	0.000	0.021	0.000	0.000	0.851

Figure 38: Normalised Confusion Matrix of the classifier trained with the augmented big dataset

C Code

```

1 #import packages we are suing for the experiment
2 import tensorflow as tf
3 import numpy as np
4 import scipy.io as io
5 from scipy.io import loadmat
6 #import skimage.measure as sk
7 import os
8 import sys
9 import h5py
10 import numpy as np
11 import scipy.io as io
12
13 import matplotlib
14 import matplotlib.pyplot as plt
15 from mpl_toolkits.mplot3d import Axes3D
16
17 from tqdm import *
18
19
20 def get_list_elements_without_pattern_not_current_directory(
21     directory_to_search):
22     #Comprehension list that by given a directory, explores all the
23     #elements
24     files= [element for element in os.listdir(directory_to_search)]
25     # return a list

```

```
24     return files
25
26 #function to create an array of an specific label
27 def create_array_labels(label,number_instances):
28     #
29     label_array=np.full((number_instances),label)
30     #
31     return label_array
32
33 def saveFromVoxels(voxels, path):
34     fig = plt.figure()
35     ax = fig.gca(projection='3d')
36     ax.voxels(voxels, facecolors='b', edgecolor='k')
37     fig.savefig(path)
38     plt.close()
39     plt.clf()
40     plt.cla()
41     del fig
42
43 def create_folder_in_path_check_folder_created(path_creation,
44                                                 path_to_create):
45     #get a list of the elements that are in the directory we want to
46     #create
47     directories_in_directory_where_eant_create=\
48     get_list_elements_without_pattern_not_current_directory(path_creation
49     )
50     #Get the paths of the elements in the directory where we want to
51     #create a
52     #new directory
53     directories_path_in_directory_where_eant_create=\
54     [path_creation+'/'+path for path in \
55     directories_in_directory_where_eant_create]
56     #If the directory we want to create has not been created before,
57     #create one
58     if path_to_create not in
59     directories_path_in_directory_where_eant_create:
60         #create the directory
61         os.mkdir(path_to_create)
62
63 def double_voxels_dimension(voxel_array_to_transform):
64     number_voxels=voxel_array_to_transform.shape[0]
65     #Get the number of voxels per dimension
66     one_dimension_voxels=voxel_array_to_transform.shape[1]
67     double_dimension=one_dimension_voxels*2
68     print (double_dimension)
69     #create an array to store the transformed instances
70     voxel_array_transformed=np.zeros((number_voxels,double_dimension,\n
71                                     double_dimension,double_dimension,1))
72
73     #loop through all the instances in the array we want to transfrom
74     for voxel_tranformation_index in range(number_voxels):
75         #get the array that we want to transform
76         voxel_to_transform=voxel_array_to_transform[
```

```
    voxel_transformation_index]
    #modify the voxels
    voxel_to_transform=np.pad(voxel_to_transform,(0, 0),\
    'constant',constant_values=(0, 0))
    #Square voxels
    voxel_to_transform=nd.zoom(voxel_to_transform,\
    (2, 2, 2), mode='constant', order=0)\.
    reshape((double_dimension,double_dimension,double_dimension,1))
    #Add the transformed voxel to the array that contains the
    transformed
    #voxels
    voxel_array_transformed[voxel_transformation_index]=\
    voxel_to_transform
    #return the modified array
    return voxel_array_transformed
84
85 #Function to reduce the size of a set of voxels
86 def half_voxels_dimension(voxel_array_to_transform):
87     number_voxels=voxel_array_to_transform.shape[0]
88     #Get the number of voxels per dimension
89     one_dimension_voxels=voxel_array_to_transform.shape[1]
90     half_dimension=int(one_dimension_voxels/2)
91
92     #create an array to store the transformed instances
93     voxel_array_transformed=np.zeros((number_voxels,half_dimension,\n94     half_dimension,half_dimension,1))
95     #loop through all the instances in the array we want to transfrom
96     for voxel_transformation_index in range(number_voxels):
97         #get the array that we want to transform
98         voxel_to_transform=voxel_array_to_transform[
99             voxel_transformation_index]
100        #modify the voxels
101        voxel_to_transform=np.pad(voxel_to_transform,(0, 0),\
102        'constant',constant_values=(0, 0))
103        #Square voxels
104        voxel_to_transform=nd.zoom(voxel_to_transform,\
105        (0.5, 0.5, 0.5), mode='constant', order=0)\.
106        .reshape((half_dimension,half_dimension,half_dimension,1))
107        #Add the transformed voxel to the array that contains the
108        transformed
109        #voxels
110        voxel_array_transformed[voxel_transformation_index]=\
111        voxel_to_transform
112    #return the modified array
113    return voxel_array_transformed
114 #
115 ######
116
117 #The below function were taken from the github post:
118 #https://github.com/meetshah1995/tf-3dgan
119
120 #Tensorflow function to create a set of weight for our models
```

```
118 def init_weights(shape, name):
119     return tf.get_variable(name, shape=shape,\n120     initializer=tf.contrib.layers.xavier_initializer())
121
122
123 #Function to create biases for the generator and discriminator
124 def init_biases(shape):
125     return tf.Variable(tf.zeros(shape))
126
127
128 #Batch normalisation layer with tensorflow
129 def batchNorm(x, n_out, phase_train, scope='bn'):
130     with tf.variable_scope(scope):
131         #Beta parameter in batch normalisation
132         beta = tf.Variable(tf.constant(0.0, shape=[n_out]), name='beta',\n133                         trainable=True)
134         #Gamma parameter
135         gamma = tf.Variable(tf.constant(1.0, shape=[n_out]), name='gamma',\n136                         trainable=True)
137         #Ema
138         batch_mean, batch_var = tf.nn.moments(x, [0,1,2], name='moments')
139         ema = tf.train.ExponentialMovingAverage(decay=0.5)
140         #function to calculate the mean value of the update
141         def mean_var_with_update():
142             ema_apply_op = ema.apply([batch_mean, batch_var])
143             with tf.control_dependencies([ema_apply_op]):
144                 return tf.identity(batch_mean), tf.identity(batch_var)
145             #
146             mean, var = tf.cond(phase_train,
147                                 mean_var_with_update,
148                                 lambda: (ema.average(batch_mean),\n149                                         ema.average(batch_var)))
150
151         #Output of as a result of the output normalisation
152         normed = tf.nn.batch_normalization(x, mean, var, beta, gamma, 1e
-3)
153
154         return normed
155
156 #Batch normalisation funtion
157 class batch_norm(object):
158     def __init__(self, epsilon=1e-5, momentum = 0.9, name="batch_norm"):
159         with tf.variable_scope(name):
160             self.epsilon = epsilon
161             self.momentum = momentum
162             self.name = name
163
164     def __call__(self, x, train=True):
165         return tf.contrib.layers.batch_norm(x, decay=self.momentum,\n166                                         updates_collections=None,\n167                                         epsilon=self.epsilon,\n168                                         scale=True,
```

```
169                     is_training=train,
170                     scope=self.name)
171
172 #Function to create a threshold for the discriminator
173 def threshold(x, val=0.5):
174     x = tf.clip_by_value(x,0.5,0.5001) - 0.5
175     x = tf.minimum(x * 10000,1)
176     return x
177
178 #LEaky relu layer
179 def lrelu(x, leak=0.2):
180     return tf.maximum(x, leak*x)
181
182
183 # def lrelu(x, leak=0.2):
184 #     f1 = 0.5 * (1 + leak)
185 #     f2 = 0.5 * (1 - leak)
186 #     return f1 * x + f2 * abs(x)
187
188 '''
189 ##### Global Parameters
190 '''
191 n_epochs    = 6000
192 batch_size  = 64
193 g_lr        = 0.0025
194 d_lr        = 0.00005
195 beta         = 0.5
196 d_thresh    = 0.8
197 z_size       = 200
198 leak_value   = 0.2
199 cube_len    = 64
200
201
202 #fucntion to create a generator
203 weights = {}
204 def generator(z, batch_size=batch_size, phase_train=True, reuse=False):
205
206     strides      = [1,2,2,2,1]
207
208     with tf.variable_scope("gen", reuse=reuse):
209         z = tf.reshape(z, (batch_size, 1, 1, 1, z_size))
210         g_1 = tf.nn.conv3d_transpose(z, weights['wg1'], (batch_size
211 ,4,4,4,512), \
212                                     strides=[1,1,1,1,1], padding="VALID")
213
214         g_1 = tf.contrib.layers.batch_norm(g_1, is_training=phase_train)
215         g_1 = tf.nn.relu(g_1)
216
217         g_2 = tf.nn.conv3d_transpose(g_1, weights['wg2'], (batch_size
218 ,8,8,8,256), \
219                                     strides=strides, padding="SAME")
220         g_2 = tf.contrib.layers.batch_norm(g_2, is_training=phase_train)
```

```

218     g_2 = tf.nn.relu(g_2)
219
220     g_3 = tf.nn.conv3d_transpose(g_2, weights['wg3'], (batch_size
221 , 16, 16, 16, 128), \
222                                     strides=strides, padding="SAME")
223     g_3 = tf.contrib.layers.batch_norm(g_3, is_training=phase_train)
224     g_3 = tf.nn.relu(g_3)
225
226     g_4 = tf.nn.conv3d_transpose(g_3, weights['wg4'], (batch_size
227 , 32, 32, 32, 64), \
228                                     strides=strides, padding="SAME")
229     g_4 = tf.contrib.layers.batch_norm(g_4, is_training=phase_train)
230     g_4 = tf.nn.relu(g_4)
231
232     g_5 = tf.nn.conv3d_transpose(g_4, weights['wg5'], (batch_size
233 , 64, 64, 64, 1), \
234                                     strides=strides, padding="SAME")
235     #Choose between an sigmoid or tanh activation function. I got
236     #the best result with the sigmoid which outputs values between 1
237     #and 0
238     g_5 = tf.nn.sigmoid(g_5)
239     #g_5 = tf.nn.tanh(g_5)
240
241     #print statements
242     print(g_1, 'g1')
243     print(g_2, 'g2')
244     print(g_3, 'g3')
245     print(g_4, 'g4')
246     print(g_5, 'g5')
247
248     return g_5
249
250 #function to generate the discriminator
251 def discriminator(inputs, phase_train=True, reuse=False):
252     #strides that the discriminator will use
253     strides = [1, 2, 2, 2, 1]
254     #Piece of code to add the multiple layers
255     with tf.variable_scope("dis", reuse=reuse):
256         d_1 = tf.nn.conv3d(inputs, weights['wd1'], strides=strides,
257                            padding="SAME")
258         d_1 = tf.contrib.layers.batch_norm(d_1, is_training=phase_train)
259         d_1 = lrelu(d_1, leak_value)
260
261         d_2 = tf.nn.conv3d(d_1, weights['wd2'], strides=strides, padding=
262 "SAME")
263         d_2 = tf.contrib.layers.batch_norm(d_2, is_training=phase_train)
264         d_2 = lrelu(d_2, leak_value)
265
266         d_3 = tf.nn.conv3d(d_2, weights['wd3'], strides=strides, padding=
267 "SAME")
268         d_3 = tf.contrib.layers.batch_norm(d_3, is_training=phase_train)
269         d_3 = lrelu(d_3, leak_value)

```

```
263
264     d_4 = tf.nn.conv3d(d_3, weights['wd4'], strides=strides, padding=
265         "SAME")
266     d_4 = tf.contrib.layers.batch_norm(d_4, is_training=phase_train)
267     d_4 = lrelu(d_4)
268
269     d_5 = tf.nn.conv3d(d_4, weights['wd5'], strides=[1,1,1,1,1],
270         padding="VALID")
271     d_5_no_sigmoid = d_5
272     d_5 = tf.nn.sigmoid(d_5)
273 #print statements
274     print(d_1, 'd1')
275     print(d_2, 'd2')
276     print(d_3, 'd3')
277     print(d_4, 'd4')
278     print(d_5, 'd5')
279
280
281 #Function to set up the initialization weights
282 def initialiseWeights():
283
284     global weights
285     xavier_init = tf.contrib.layers.xavier_initializer()
286
287     weights['wg1'] = tf.get_variable("wg1", shape=[4, 4, 4, 512, 200],
288         initializer=xavier_init)
289     weights['wg2'] = tf.get_variable("wg2", shape=[4, 4, 4, 256, 512],
290         initializer=xavier_init)
291     weights['wg3'] = tf.get_variable("wg3", shape=[4, 4, 4, 128, 256],
292         initializer=xavier_init)
293     weights['wg4'] = tf.get_variable("wg4", shape=[4, 4, 4, 64, 128],
294         initializer=xavier_init)
295     weights['wg5'] = tf.get_variable("wg5", shape=[4, 4, 4, 1, 64],
296         initializer=xavier_init)
297
298     weights['wd1'] = tf.get_variable("wd1", shape=[4, 4, 4, 1, 64],
299         initializer=xavier_init)
300     weights['wd2'] = tf.get_variable("wd2", shape=[4, 4, 4, 64, 128],
301         initializer=xavier_init)
302     weights['wd3'] = tf.get_variable("wd3", shape=[4, 4, 4, 128, 256],
303         initializer=xavier_init)
304     weights['wd4'] = tf.get_variable("wd4", shape=[4, 4, 4, 256, 512],
305         initializer=xavier_init)
306     weights['wd5'] = tf.get_variable("wd5", shape=[4, 4, 4, 512, 1],
307         initializer=xavier_init)
308
309     return weights
310
311 #Function to ensemble the gan and train it
312 def trainGAN(dataset, path, label, is_dummy=False, checkpoint=None):
313     #'content/drive/My Drive/working/merged_data/analysis_gans_final/full
```

```
dataset/3/biasfree_350.cptk'):

304     weights = initialiseWeights()

305
306
307     z_vector = tf.placeholder(shape=[batch_size, z_size], dtype=tf.float32)
308     x_vector = tf.placeholder(shape=[batch_size, cube_len, cube_len,
309                               cube_len, 1], \
310                               dtype=tf.float32)

311     net_g_train = generator(z_vector, phase_train=True, reuse=False)

312
313     d_output_x, d_no_sigmoid_output_x = discriminator(x_vector,
314                                                       phase_train=True, \
315                                                       reuse=False)
316     d_output_x = tf.maximum(tf.minimum(d_output_x, 0.99), 0.01)
317     summary_d_x_hist = tf.summary.histogram("d_prob_x", d_output_x)

318     d_output_z, d_no_sigmoid_output_z = discriminator(net_g_train,
319                                                       phase_train=True, reuse=True)
320     d_output_z = tf.maximum(tf.minimum(d_output_z, 0.99), 0.01)
321     summary_d_z_hist = tf.summary.histogram("d_prob_z", d_output_z)

322     # Compute the discriminator accuracy
323     n_p_x = tf.reduce_sum(tf.cast(d_output_x > 0.5, tf.int32))
324     n_p_z = tf.reduce_sum(tf.cast(d_output_z < 0.5, tf.int32))
325     d_acc = tf.divide(n_p_x + n_p_z, 2 * batch_size)

326
327     # Compute the discriminator and generator loss
328     # d_loss = -tf.reduce_mean(tf.log(d_output_x) + tf.log(1-d_output_z))
329     # g_loss = -tf.reduce_mean(tf.log(d_output_z))

330
331     d_loss = tf.nn.sigmoid_cross_entropy_with_logits(logits=
332                                                       d_no_sigmoid_output_x, \
333                                                       labels=tf.ones_like(
334                                                       d_output_x))
335     d_loss += tf.nn.sigmoid_cross_entropy_with_logits(logits=
336                                                       d_no_sigmoid_output_z, \
337                                                       labels=tf.
338                                                       zeros_like(d_output_z))
339     g_loss = tf.nn.sigmoid_cross_entropy_with_logits(logits=
340                                                       d_no_sigmoid_output_z, \
341                                                       labels=tf.ones_like(
342                                                       d_output_z))

343     d_loss = tf.reduce_mean(d_loss)
344     g_loss = tf.reduce_mean(g_loss)

345
346     summary_d_loss = tf.summary.scalar("d_loss", d_loss)
347     summary_g_loss = tf.summary.scalar("g_loss", g_loss)
348     summary_n_p_z = tf.summary.scalar("n_p_z", n_p_z)
349     summary_n_p_x = tf.summary.scalar("n_p_x", n_p_x)
350     summary_d_acc = tf.summary.scalar("d_acc", d_acc)
```

```
347     net_g_test = generator(z_vector, phase_train=False, reuse=True)
348
349     para_g = [var for var in tf.trainable_variables() if any(x in var.
350 name for x in ['wg', 'bg', 'gen'])]
350     para_d = [var for var in tf.trainable_variables() if any(x in var.
351 name for x in ['wd', 'bd', 'dis'])]
352
352     # only update the weights for the discriminator network
353     optimizer_op_d = tf.train.AdamOptimizer(learning_rate=d_lr, beta1=beta
353 ).minimize(d_loss, var_list=para_d)
354     # only update the weights for the generator network
355     optimizer_op_g = tf.train.AdamOptimizer(learning_rate=g_lr, beta1=beta
355 ).minimize(g_loss, var_list=para_g)
356
356     saver = tf.train.Saver()
357
358
359     with tf.Session() as sess:
360
361         sess.run(tf.global_variables_initializer())
362         #Load checkpoints in case we need to retrain our model
363         if checkpoint is not None:
364             saver.restore(sess, checkpoint)
365
366         if is_dummy:
367             volumes = np.random.randint(0,2,(batch_size,cube_len,cube_len
367 ,cube_len))
368             print('Using Dummy Data')
369         else:
370             volumes = dataset.astype(np.float)
371             print ('using own data')
372             # volumes *= 2.0
373             # volumes -= 1.0
374
375             #Lists to keep track of the loss fucntions and accuracies of the
375             #generator
376             #and discriminator
377             loss_function_generator=[]
378             loss_function_discriminator=[]
379             discriminator_accuracy=[]
380
381
382             for epoch in range(n_epochs):
383
384                 idx = np.random.randint(len(volumes), size=batch_size)
385                 x = volumes[idx]
386                 z_sample = np.random.normal(0, 0.33, size=[batch_size, z_size
386 ]).astype(np.float32)
387                 z = np.random.normal(0, 0.33, size=[batch_size, z_size]).as
387 type(np.float32)
388                 # z = np.random.uniform(0, 1, size=[batch_size, z_size]).as
388 type(np.float32)
389
390                 # Update the discriminator and generator
```

```
391     d_summary_merge = tf.summary.merge([summary_d_loss,
392                                         summary_d_x_hist,
393                                         summary_d_z_hist,
394                                         summary_n_p_x,
395                                         summary_n_p_z,
396                                         summary_d_acc])
397
398     summary_d, discriminator_loss = sess.run([d_summary_merge,
399                                               d_loss], feed_dict={z_vector:z, x_vector:x})
400     summary_g, generator_loss = sess.run([summary_g_loss, g_loss],
401                                           feed_dict={z_vector:z})
402     d_accuracy, n_x, n_z, d_x,d_z = sess.run([d_acc, n_p_x, n_p_z,
403                                               d_output_x,d_output_z], feed_dict={z_vector:z, x_vector:x})
404
405     #print("nx_nz:",n_x, n_z, "\nd_x:",d_x.reshape(batch_size), "d_z:",
406           d_z.reshape(batch_size))
407
408     print ("nx",n_x,"nz",n_z)
409     if d_accuracy < d_thresh:
410         sess.run([optimizer_op_d], feed_dict={z_vector:z, x_vector:
411 :x})
412         print('Discriminator Training ', "epoch: ",epoch,',
413               d_loss:',discriminator_loss,'g_loss:',generator_loss, "d_acc: ",
414               d_accuracy)
415
416     sess.run([optimizer_op_g], feed_dict={z_vector:z})
417
418     #Append values to the lists that keep track of the results
419     loss_function_generator.append(generator_loss)
420     loss_function_discriminator.append(discriminator_loss)
421     discriminator_accuracy.append(d_accuracy)
422
423     print('Generator Training ', "epoch: ",epoch,',
424           d_loss:',discriminator_loss,'g_loss:',generator_loss, "d_acc: ",
425           d_accuracy)
426
427 ##### END OF THE CODE retrieve from https://github.com/
428 ##### meetshah1995/tf-3dgan #####
429
430     #Print generations and store generated data
431     if epoch % 50 == 0:
432         #generate data
433         voxel_volumes= sess.run(net_g_test,feed_dict={z_vector:
434 z_sample})
435         z = np.random.normal(0, 0.33, size=[batch_size, z_size]).\
436         astype(np.float32)
437         voxels_volumens_II= sess.run(net_g_test,feed_dict={
438 z_vector:z})
439         #get random numbers as index to retrieve generated
440         instances
441         id_ch = np.random.randint(0, batch_size, 4)
442
443         #plot the random generated data
444         for i in range(3):
```

```
431         print(voxel_volumes[id_ch[i]].max())
432         if voxel_volumes[id_ch[i]].max() > 0.5:
433             voxels = np.squeeze(voxel_volumes[id_ch[i]])
434             #filter the voxels to binary values
435             voxels[voxels < 0.5] = 0
436             voxels[voxels >= 0.5] = 1
437             #modify the shape of the voxels
438             voxels=nd.zoom(voxels,\n(0.5, 0.5, 0.5), mode='constant', order=0)\\
439             .reshape((32,32,32))
440             #save image
441             saveFromVoxels(voxels, path+"/img_{0}_{1}".format(epoch,
442 i))
443
444             #concatenate arrays
445             generated_array=np.concatenate((voxel_volumes,
446 voxels_volumens_II),axis=0)
447             del voxel_volumes
448             del voxels_volumens_II
449
450             #get information of our generations and create a label
451             array
452             number_generated_instances=generated_array.shape[0]
453             label_array=create_array_labels(label,
454             number_generated_instances)
455
456             #save the array
457             with h5py.File(path+"/generated_data_array_{0}".format(
458 epoch)+'.h5', 'w') as hf:
459                 hf.create_dataset("generated_data",data=generated_array
460 )
461                 hf.create_dataset("label",data=label_array)
462                 hf.close()
463
464             #delete the generated array
465             del generated_array
466
467             #save a checkpoint of the model to load it to generate extra
468             data or
469             #to keep going with the training in case of interruption
470             if epoch % 50 == 0:
471                 saver.save(sess, save_path = path + '/biasfree_' + str(
472 epoch)+'.cptk')
473
474             #save the evolution of the loss functions
475             with h5py.File(path+"/evolution_loss_functions{0}".format(
476 epoch)+'.h5', 'w') as hf:
477                 hf.create_dataset("generator_loss",data=
478 loss_function_generator)
479                 hf.create_dataset("discriminator_loss",data=
480 loss_function_discriminator)
481                 hf.create_dataset("discriminator_accuracy",data=
482 discriminator_accuracy)
```

```
472         hf.close()
473
474     def generateGAN(path, label, trained_model_path=None, epoch='last',
475                     n_batches=10):
476         weights = initialiseWeights()
477
478         z_vector = tf.placeholder(shape=[batch_size, z_size], dtype=tf.float32)
479         net_g_test = generator(z_vector, phase_train=True, reuse=False)
480
481         sess = tf.Session()
482         saver = tf.train.Saver()
483
484         with tf.Session() as sess:
485             sess.run(tf.global_variables_initializer())
486             saver.restore(sess, trained_model_path)
487
488             #generate data
489             #
490             for i in range(n_batches):
491                 if i == 0:
492                     #next_sigma = float(raw_input())
493                     z_sample = np.random.normal(0, 0.33, size=[batch_size, z_size])
494                     .astype(np.float32)
495                     generated_data=sess.run(net_g_test,feed_dict={z_vector:z_sample})
496
497                 else:
498                     #next_sigma = float(raw_input())
499                     z_sample = np.random.normal(0, 0.33, size=[batch_size, z_size])
500                     .astype(np.float32)
501                     generated_samples=sess.run(net_g_test,feed_dict={z_vector:
502 z_sample})
503                     generated_data=np.concatenate((generated_data,
504                     generated_samples), axis=0)
505
506             number_generated_instances=generated_data.shape[0]
507             label_array=create_array_labels(label, number_generated_instances)
508
509             #generate the dataset
510             with h5py.File(path+"/generated_data_array_{0}_{1}.format(epoch, str(
511                             label))+'.h5', 'w') as hf:
512                 hf.create_dataset("generated_data",data=generated_data)
513                 hf.create_dataset("labels",data=label_array)
514             hf.close()
515
516     def train_multiple_dataset_with_multiple_labels(list_data_sets_paths, \
517                                                   list_dataset_names, labels, root_directory):
518         number_datasets_to_analyse=len(list_data_sets_paths)
519         number_labels_analysis=len(labels)
520         for dataset_index in range(list_data_sets_paths):
521             #get the name oof the dataset we are analysis
522             dataset_analysis=list_data_sets_paths[dataset_index]
```

```
518     model=list_dataset_names[dataset_index]
519     for label_index in range(number_labels_analysis):
520         #get the labels of analysis
521         label=labels[label_index]
522
523         #creare directories to store results and get the data
524         current_directory=root_directory
525         results_directory=current_directory+='/gans_results'
526         dataset_directory=results_directory+'/' +str(model)
527         label_directory=dataset_directory+'/' +str(label)
528         #Create directories
529         create_folder_in_path_check_folder_created(current_directory,
530             results_directory)
530         create_folder_in_path_check_folder_created(results_directory,
531             dataset_directory)
532         create_folder_in_path_check_folder_created(dataset_directory,
533             label_directory)
534         #
535
536         #get the data
537         dataset= h5py.File(dataset_analysis, 'r')
538         attributes_training=np.array(dataset.get('attributes_training'))
539     )
540         attributes_testing=np.array(dataset.get('attributes_testing'))
541     )
542         labels_training=np.array(dataset.get('labels_training'))
543         labels_testing=np.array(dataset.get('labels_testing'))
544         dataset.close()
545         del dataset
546         del attributes_testing
547
548         #transform the data
549         boolean_mask=np.where(labels_training == label)[0]
550
551         #Get the instances that correspond with the labels
552         gans_data=attributes_training[boolean_mask][:500]
553         gans_data=gans_data.reshape((-1,32,32,32))
554
555         #Transform the data to 64x64x64 format
556         gans_data=double_voxels_dimension(gans_data)
557
558         #Training process
559         trainGAN(gans_data,label_directory,label)
560
561 ##### END FUNCTIONS #####
562
563 #Create the first and second models
564 #list_data_sets_paths=['merged_dataset_0.2labelled_instances.h5', \
565 #'merged_dataset_0.4labelled_instances.h5','merged_dataset_0.6 \
566   labelled_instances.h5', \
567 #'merged_dataset_0.8labelled_instances.h5','merged_dataset.h5']
```

```

564 list_data_sets_paths=['merged_dataset_0.2labelled_instances.h5', \
565 'merged_dataset_0.4labelled_instances.h5', 'merged_dataset_0.6 \
      labelled_instances.h5', \
566 'merged_dataset_0.8labelled_instances.h5', 'merged_dataset.h5']
567
568 #name of the models we are going to use
569 #list_dataset_names=['0.20 dataset','0.40 dataset','0.60 dataset','0.80 \
      dataset', \
570 #'full dataset']
571 list_dataset_names=['0.20 dataset','0.40 dataset','0.60 dataset','0.80 \
      dataset', \
572 'full dataset']
573
574 #labels of the dataset we are going to use
575 #labels=[0,1,2,3,4,5,6,7,8,9,10,11,12,13]
576 labels=[0,1,2,3,4,5,6,7,8,9,10,11,12,13]
577
578 #train_multiple_dataset_with_multiple_labels(list_data_sets_paths, \
579 #list_dataset_names,labels,root_directory)
580
581 path_check_check_point='G:/gans_project_root_directory/processed_data/\ \
582 gans_results/1/checkpoints_and_arrays/biasfree_3950.cptk'
583
584 result_generation='G:/gans_project_root_directory/processed_data/\ \
      gans_results/1/new_generated_data'
585
586 generateGAN(result_generation, 1, trained_model_path= \
      path_check_check_point, \
      epoch='3900', n_batches=10)

```

Listing 1: 3D GANs code

```

1 #load utils from keras
2 #Import Keras tools we use to implements GANs
3 import tensorflow as tf
4 from tensorflow import keras
5
6 #Load utils from skleanr
7 from sklearn.metrics import confusion_matrix, accuracy_score
8 from sklearn.model_selection import train_test_split
9
10 #Load utils from standard libraries
11 import h5py
12 import pandas as pd
13 import numpy as np
14 import matplotlib.pyplot as plt
15 from matplotlib import cm
16 import seaborn as sns
17 sns.set_style('white')
18
19
20 class IIID_classification():
21     def __init__(self):

```

```
23     self.horizontal_axis=16
24     self.vertical_axis=16
25     self.volume_axis=16
26     self.color_channels=3
27     self.input_size=(self.horizontal_axis,self.vertical_axis,\n
28     self.volume_axis, self.color_channels)
29     self.number_classes=10
30     self.one_dimension_size=4096
31
32     #Training parameters Good combinations:(30,80),
33     self.epochs=2
34     self.batch=86
35     #batch_size=128, epochs=50
36     self.validation_split=0.20
37     self.learning_rate=0.001
38
39     #Normally 3D model are in h5 format.Open h5 files and separate the
40     instances
41     #within them into training and testing files.
42     def open_h5(self,file_to_open='aaa'):
43         with h5py.File(file_to_open+".h5", 'r') as h5:
44             attributes_training,labels_training=h5["X_train"][:,],h5["\n
45             y_train"][:,]
46             attributes_testing,labels_testing= h5["X_test"][:,], h5["\n
47             y_test"][:,]
48
49             return attributes_training,labels_training,attributes_testing
50             ,\n
51             labels_testing
52
53     #In most of the datasets the 3D data is in 1D. So, we have to
54     process
55     #this data for its visualization and posterio analysis
56
57     #Find the rgb values of our dataset
58     def add_rgb_dimention(self, instance):
59         #Choose the color map we are using
60         scaler_map = cm.ScalarMappable(cmap="Oranges")
61
62         #Transform the instance. The -1 fits automatically the size to
63         #the
64         #dimension
65         instance= scaler_map.to_rgba(instance)[:, : -1]
66
67         return instance
68
69     #Process to transform our 1D data to 3D data
70     def add_color_dimension(self, dataset_to_transform):
71         dataset_with_color_coordinates=np.ndarray((\
72         dataset_to_transform.shape[0],self.one_dimension_size,3))
73
74         #Loop through all the instance to add the color coordinates
75         for instance_index in range(dataset_to_transform.shape[0]):
```

```
70         dataset_with_color_coordinates[instance_index]=\
71             self.add_rgb_dimention(dataset_to_transform[instance_index])
72
73     return dataset_with_color_coordinates
74
75 def reshape_dataset(self , dataset):
76     #convert our data set to a 'number of instance' + 4D dimensional
77     #dataset
78     # the '-1' automatically calculates the remaining dimension.
79     dataset = dataset.reshape(-1,self.horizontal_axis , self.
80     vertical_axis,\n        self.volume_axis , self.color_channels)
81
82     return dataset
83
84 def one_hot_encode_labels(self ,labels):
85     #convert target variable into one-hot
86     labels = keras.utils.to_categorical(labels ,self.number_classes)
87
88     return labels
89
90 def Conv(self,filters=16, kernel_size=(3,3,3),activation='relu',\
91 input_shape=None):
92     if input_shape:
93         return keras.layers.Conv3D(filters=filters,kernel_size=
94     kernel_size,\n            padding='Same' , activation=activation , input_shape=
95     input_shape)
96     else:
97         return keras.layers.Conv3D(filters=filters,kernel_size=
98     kernel_size,\n            padding='Same' , activation=activation)
99
100 #3D convolutional networks require a tensor innput of five dimensions
101 #:
102 #number of instances per batch, horizontal dimension , vertical
103 #dimension ,
104 #volumen dimension , number of color channels.
105 def convolutional_IIID_network(self):
106     #Common structure of 3CNN
107     ## input layer
108     input_layer= keras.layers.Input((self.input_size))
109
110     ## Add the 3D convolutional layers with different characteristics
111     #The parenthesis after the layer connect the previous layer with
112     #the layer we have already created.
113     conv_layer1 = keras.layers.Conv3D(filters=8, kernel_size=(3, 3,
114     3),\n            activation='relu')(input_layer)
115
116     #Add more 3D convolutional layers and 3D maxpool layers.
117     conv_layer2 = keras.layers.Conv3D(filters=16, kernel_size=(3, 3,
```

```
115     3), \
116         activation='relu')(conv_layer1)
117
118     ## add max pooling to obtain the most imformatic features
119     pooling_layer1 = keras.layers.MaxPool3D(pool_size=(2, 2, 2))\
120     (conv_layer2)
121
122     conv_layer3 = keras.layers.Conv3D(filters=32, kernel_size=(3, 3,
123 ), \
124         activation='relu')(pooling_layer1)
125
126     conv_layer4 = keras.layers.Conv3D(filters=64, kernel_size=(3, 3,
127 ), \
128         activation='relu')(conv_layer3)
129     pooling_layer2 = keras.layers.MaxPool3D(pool_size=(2, 2, 2))\
130     (conv_layer4)
131
132     #perform batch normalization on the convolution outputs before
133     #feeding
134     #it to MLP architecture
135     pooling_layer2 = keras.layers.BatchNormalization()(pooling_layer2)
136 )
137     flatten_layer = keras.layers.Flatten()(pooling_layer2)
138
139     #Fully connected layer of top of the convolutions to classify the
140     #model
141     #First transform the results of the convolution into a 1D format
142     dense_layer1 = keras.layers.Dense(units=2048, activation='relu')\
143     (flatten_layer)
144     dense_layer1 = keras.layers.Dropout(0.4)(dense_layer1)
145     dense_layer2 = keras.layers.Dense(units=512, activation='relu')\
146     (dense_layer1)
147     dense_layer2 = keras.layers.Dropout(0.4)(dense_layer2)
148     output_layer = keras.layers.Dense(units=self.number_classes, \
149         activation='softmax')\
150     (dense_layer2)
151
152     ## define the model with input layer and output layer
153     model=keras.models.Model(inputs=input_layer, outputs=output_layer)
154 )
155
156     #Compile the model
157     model.compile(loss="categorical_crossentropy", \
158         optimizer=keras.optimizers.Adadelta(lr=0.1),metrics=[ "accuracy"])
159
160     return model
161
162 def convolutional_IID_network_other_Structure(self):
163     #Normal feed-forward structure
164     cnn_three=keras.models.Sequential()
165
166     #USe the fucntion Conv that we create before to cast the 3D
167     #convolutional network
```

```
161     cnn_three.add(self.Conv(8, (3,3,3), input_shape=self.input_size))
162     cnn_three.add(self.Conv(16, (3,3,3)))
163
164     #model.add(BatchNormalization())
165     cnn_three.add(keras.layers.MaxPool3D())
166     #cnn_three.add(keras.layers.Dropout(0.25))
167
168     #
169     cnn_three.add(self.Conv(32, (3,3,3)))
170     cnn_three.add(self.Conv(64, (3,3,3)))
171     cnn_three.add(keras.layers.BatchNormalization())
172     cnn_three.add(keras.layers.MaxPool3D())
173     cnn_three.add(keras.layers.Dropout(0.25))
174
175     #Fully connected layer of top of the convolutions to classify the
176     #model
177     #First transform the results of the convolution into a 1D format
178     cnn_three.add(keras.layers.Flatten())
179
180     #Add more fully connected layers
181     cnn_three.add(keras.layers.Dense(4096, activation='relu'))
182     cnn_three.add(keras.layers.Dropout(0.5))
183     cnn_three.add(keras.layers.Dense(1024, activation='relu'))
184     cnn_three.add(keras.layers.Dropout(0.5))
185     #The input layer contains as many as neurons as different classes
186     #cnn_three.add(keras.layers.Dense(self.number_classes,activation='softmax'))
187
188     #Compile the model
189     cnn_three.compile(optimizer='adam',loss =
190                         "categorical_crossentropy",
191                         metrics=["accuracy"])
192
193     #return the model
194     return cnn_three
195
196 def voxnet(self):
197     #Common structure of 3CNN
198     ## input layer
199     input_layer= keras.layers.Input((self.input_size))
200     ## Add the 3D convolutional layers with different characteristics
201     #The parenthesis after the layer connect the previous layer with
202     #the layer we have already created.
203     conv_layer1 = keras.layers.Conv3D(filters=32, kernel_size=(5,5,5),
204                                     strides=(2,2,2),activation='relu')(input_layer)
205
206     #Add more 3D convolutional layers and 3D maxpool layers.
207     conv_layer2 = keras.layers.Conv3D(filters=32, kernel_size=(3, 3,
208                                     3),strides=(1,1,1),activation='relu')(conv_layer1)
209
210     ## add max pooling to obtain the most imformatic features
```

```
209     pooling_layer1 = keras.layers.MaxPool3D(pool_size=(2, 2, 2))\
210     (conv_layer2)
211     #
212     flatten_layer = keras.layers.Flatten()(pooling_layer1)
213
214     #Fully connected layer of top of the convolutions to classify the
215     #model
216     #First transform the results of the convolution into a 1D format
217     dense_layer1 = keras.layers.Dense(units=128, activation='relu')\
218     (flatten_layer)
219     dense_layer1 = keras.layers.Dropout(0.5)(dense_layer1)
220     output_layer = keras.layers.Dense(units=self.number_classes,\n
221     activation='softmax')(dense_layer1)
222
223     ## define the model with input layer and output layer
224     model=keras.models.Model(inputs=input_layer,outputs=output_layer)
225
226     #Compile the model
227     model.compile(loss="categorical_crossentropy",\
228     optimizer=keras.optimizers.SGD(lr=self.learning_rate,momentum
229     =0.9),\
230     metrics=["accuracy"])
231
232     return model
233
234     #Function to create a callback to stop the training process given a
235     #set
236     #of characteristics
237     def generate_stopping_criteria(self, monitor_metric='val_accuracy',\
238     callback_patience=20):
239         #create the callback object to stop the training process.
240         Patience
241         #is the number of iterations without improvement that have to
242         happen to
243         #stop the training process. Monitor is the performance measure
244         #that we
245         #consider to stop the training process.
246         stopping_call_back=keras.callbacks.EarlyStopping(monitor=
247         monitor_metric,\n
248         mode='max',verbose=1,patience=callback_patience)
249         #return the callback
250         return stopping_call_back
251
252     #Function to save our model given a specific criteria
253     def generate_model_saving_criteria(self, monitor_metric='val_accuracy'
254     ,\
255     model_name='best_model'):
256         #generate hte name of the h5 that will store the best model
257         model=model_name+'.h5'
258         #generate the callback to store the best model. The monitor
259         metric
260         #is the measure that we want to maximize with out model
261         saving_call_back=keras.callbacks.ModelCheckpoint(model,\n
```

```
253     monitor=monitor_metric, mode='max', verbose=1, save_best_only=True)
254     #return the call back
255     return saving_call_back
256
257     # Train model with the parametrs indicated in the constructor
258     def train_model(self, model_to_train, attributes_training,
259                     labels_training,\n                      callback_list):
260         #Train the model
261         train_model=model_to_train.fit(x=attributes_training,y=
262                                         labels_training,\n                                         batch_size=self.batch, epochs=self.epochs,\n                                         validation_split=self.validation_split, verbose=1, shuffle=True,\n                                         callbacks=callback_list)
263         #Return the trained model
264         return train_model
265
266
267
268     #The following fucntion saves the model that we are training.
269     def save_smodel(self, model_to_save):
270         saved_model=model_to_save.save('3d_classifier.h5')
271
272
273     #Evaluate the model after the training process.
274     def model_evaluation(self, model_to_evaluate, attributes_testing,\n                          labels_testing, path='path',title='confusion_matrix'):
275         #Predict the labels using the model we trained
276         class_prediction=model_to_evaluate.predict(attributes_testing)
277         #Because the model is one hot encoded we have and we used softmax
278         #activation fucntion
279         class_prediction=np.argmax(class_prediction, axis=1)
280
281         #Calculate the accuracy score of our model.
282         accuracy_model=round(accuracy_score(class_prediction,
283                                     labels_testing),3)
284
285         #Confusin matrix. The confucion matrix will indicate which labels
286         #are
287         #hard to predict and other potential problem in out model.
288         confusion_matrix_model=confusion_matrix(labels_testing,
289                                               class_prediction)
290
291         #transform the numpy array to a pandas dataframe
292         confusion_matrix_model=pd.DataFrame(confusion_matrix_model,\n                                             index = range(self.number_classes),\n                                             columns = range(self.number_classes)).astype('int')
293
294         #Plot the confusion matrix
295         plt.figure(figsize=(20,20))
296         sns.heatmap(confusion_matrix_model, annot=True)
297         plt.title(title+' '+'accuracy: '+str(accuracy_model))
298         plt.savefig(path+'.png')
299         plt.clf()
300         #return the model evaluation object
301         return accuracy_model
```

```
301 #Plot the evolution of the training process
302 def plot_validation_score(self, model, path='path', \
303 title='validation_training'):
304     #We are going to plot the training and validation scores. We will
305     #create
306     #a .png create the figure we are going to plot our accuracy and
307     #the
308     #value of the loss function
309     plt.figure(figsize=[20,20])
310     #Plot the accuracy evolution
311     plt.plot(model.history['accuracy'])
312     plt.plot(model.history['val_accuracy'])
313     plt.title('Model training and validation_'+title)
314     plt.ylabel('performance')
315     plt.xlabel('epoch')
316     plt.legend(['training set','validation set'], loc='upper left')
317
318     #save figure we created
319     plt.savefig(path+'.png')
320     plt.clf()
321
322 #The following function is to test the performance of our model in
323 #different
324 #set of our training set with different sizes.
325 #This could be useful to evaluate the performance of the model in
326 #when
327 #we have small datasets.
328 #list_of_models is a list that contains the models we are using
329 ##list_of_models is a list that contains the splits or reduction of
330 #the
331 #dataset we are using
332 def dataframe_record_experiment_results(self,list_of_models,
333 list_of_splits,\n334 attributes_training,labels_training,attributes_testing,labels_testing
335 ,\
336 number_data_splits=3,random_seed=0):
337     #Get the number of models
338     number_models=len(list_of_models)
339
340     #Get the dataset reduction we want to apply
341     number_data_splits=len(list_of_splits)
342
343     #create an array to store the results
344     results_array=np.zeros((number_data_splits,number_models))
345
346     #loop through the models and
347     for model_index in range(number_models):
348         #loop through the splits
349         for split_index in range(number_data_splits):
350             #Know the model we are using and how much are we gonna
351             #reduce
352             #the dataset
```

```

346     data_division=list_of_splits[split_index]
347     model=list_of_models[model_index]
348
349     #reduce the dataset
350     attributes_reduced, attributes_discard,\n
351     labels_reduced, labels_discard = train_test_split(
352     attributes_training, labels_training,\n
353     test_size=data_division, random_state=random_seed)
354
355     #Train, evaluate and plot the model
356     model_train= self.\n
357     train_model(model, attributes_reduced, labels_reduced)
358
359     #Evaluate the model
360     performance=self.model_evaluation(model,\n
361     attributes_testing,labels_testing,\n
362     title='confusion_matrix_+'+model_+str(model_index)+'_'+\
363     'used_data_'+str(data_division))
364
365     #Plot the evoluation of the validation score and the
366     #training
367     #accuracy
368     self.plot_validation_score(model_train,\n
369     file_name='validation_training'+model_+str(model_index)\n
370     +'_'+\
371     'used_data_'+str(data_division))
372
373     results_array[split_index,model_index]=performance
374
375     #Issue a csv file with the result of our model in different data
376     #reductions
377     pd.DataFrame(results_array).to_csv('results.csv')
378
379     return results_array

```

Listing 2: 3D Deep Learning classifiers

```

1 #load utils from keras
2 #Import Keras tools we use to implements GANs
3 import tensorflow as tf
4 from tensorflow import keras
5
6 #Load utils from skleanr
7 from sklearn.metrics import confusion_matrix, accuracy_score
8 from sklearn.model_selection import train_test_split
9
10 #Load utils from standard libraries
11 import h5py
12 import math
13 import pandas as pd
14 import numpy as np
15 import matplotlib.pyplot as plt
16 from matplotlib import cm
17 import seaborn as sns

```

```
18 sns.set_style('white')
19 import os
20
21 #import the code we created
22 from IIId_classifiers import IIId_classification
23
24 #
##########
25 #function to create folders in a given path
26
27 def get_list_elements_without_pattern_not_current_directory(
28     directory_to_search):
29     #Comprehension list that by given a directory, explores
30     files= [element for element in os.listdir(directory_to_search)]
31     #
32     return files
33
34
35 def create_folder_in_path_check_folder_created(path_creation,
36     path_to_create):
37     #
38     directories_in_directory_where_eant_create=\
39     get_list_elements_without_pattern_not_current_directory(path_creation)
39     #
40     directories_path_in_directory_where_eant_create=\
41     [path_creation+'/'+path for path in \
42     directories_in_directory_where_eant_create]
43     #
44     if path_to_create not in
45     directories_path_in_directory_where_eant_create:
46         #
47         os.mkdir(path_to_create)
48
#####
49
50 #Load the dataset
51 def analysis_data_and_class_creation(data_set_name, epochs=1000, batchs
52 =32,\n52 validation=0.50, learning_rate=0.001):
53     #Open the h5 file with the function within the class
54     dataset= h5py.File(data_set_name, 'r')
55     attributes_training=np.array(dataset.get('attributes_training'))
56     attributes_testing=np.array(dataset.get('attributes_testing'))
57     labels_training=np.array(dataset.get('labels_training'))
58     labels_testing=np.array(dataset.get('labels_testing'))
59     dataset.close()
60
61     #Transform the dataset
```

```
62     number_voxels_columns=attributes_training.shape[1]
63     #
64     dimension_axis=int(round(math.pow(number_voxels_columns,1/3.)))
65     #
66     attributes_training=attributes_training.reshape(-1,dimension_axis,\ 
67     dimension_axis,dimension_axis,1)
68     #
69     attributes_testing=attributes_testing.reshape(-1,dimension_axis,\ 
70     dimension_axis,dimension_axis,1)
71     #
72     number_different_labels=len(np.unique(labels_training))
73
74     #Load the class
75     IIID_classifier= IIID_classification()
76     IIID_classifier.horizontal_axis=dimension_axis
77     IIID_classifier.vertical_axis=dimension_axis
78     IIID_classifier.volume_axis=dimension_axis
79     IIID_classifier.color_channels=1
80     IIID_classifier.input_size=(IIID_classifier.horizontal_axis,\ 
81     IIID_classifier.vertical_axis,IIID_classifier.volume_axis,\ 
82     IIID_classifier.color_channels)
83     IIID_classifier.number_classes=number_different_labels
84     IIID_classifier.one_dimension_size=number_voxels_columns
85
86     #Training parameters Good combinations:(30,80),
87     IIID_classifier.epochs=epochs
88     IIID_classifier.batch=batchs
89     #batch_size=128, epochs=50
90     IIID_classifier.validation_split=validation
91     IIID_classifier.learning_rate=learning_rate
92
93     #One hot encode training and testing labels
94     labels_training=labels = keras.utils.to_categorical(labels_training)
95     #return the classifier object
96     return IIID_classifier
97
98
99 ##### TRAINING
100 ##### TRAIN MODEL I
101
102 #Function to analyse a large number of dataset and store the results of a
103 #chosen
104 #model into a csv. Hence, evaluate the performance of the model in
105 #multiple
106 #datasets. Additionally we can repeat the evaluation process several times
107 #to
108 #calculate the average and std measure of the performance.
109 def multiple_data_model_analysis(list_datasets, list_titles, list_models
110     ,\
111     number_analysis_per_dataset=5,patience=50, csv_title='csv_multiple_data')
112     :
```

```
108     #create a directory to save the results
109     #get current directory
110     current_directory=os.getcwd()
111     #directory we will create
112     directory_results=current_directory+ '/' +'results_analysis_datasets'
113     #create the dataset
114     create_folder_in_path_check_folder_created\
115     (current_directory,directory_results)
116
117     #Basic analysis of the number of data we have to analyse
118     number_sets_to_analyse=len(list_data_sets)
119     number_model_analysis=len(model_names)
120
121     #Create an array to store the results. Every column
122     #is a different dataset
123     results_array=np.zeros((number_model_analysis*4,
124     number_sets_to_analyse))
125
126     #Create and 3d classifier object with the characteristics of our data
127     IID_classifier=analysis_data_and_class_creation(\n
128     list_data_sets[number_model_analysis-1],epochs=1000, batchs=32,\n
129     validation=0.50,learning_rate=0.001)
130
131     #Loop through all the datasets
132     for dataset_index in range(number_sets_to_analyse):
133         #Get the dataset of analysis
134         data_set_of_analysis=list_data_sets[dataset_index]
135         #Get the title/label of the dataset we are analysing.
136         #The title will appear on the confusion matrix, table of
137         #results and other visualizations.
138         title_data_analysis=titles[dataset_index]
139
140         print 'dataset: '+' '+data_set_of_analysis+' '+
141         title_data_analysis
142
143         #create a numpy array to store the results of the model
144         #on the dataset after a number of repetitions
145         results_model_array=np.zeros(number_analysis_per_dataset)
146         results_model_accuracy_array=np.zeros(number_analysis_per_dataset)
147     )
148
149     #repeat the analysis as indicated in 'number_analysis_per_dataset'
150     ,
151
152     for analysis_index in range(number_analysis_per_dataset):
153         print 'analysis_number: '+str(analysis_index)
154         #load the classification model
155         IID_model_I=IID_classifier.voxnet()

156         #load the data from dataset
157         dataset= h5py.File(data_set_of_analysis,'r')
158         attributes_training=np.array(dataset.get('attributes_training
159         '))
160         attributes_testing=np.array(dataset.get('attributes_testing'))
```

```
    )
156     labels_training=np.array(dataset.get('labels_training'))
157     labels_testing=np.array(dataset.get('labels_testing'))
158     dataset.close()
159
160     #Transform the dataset
161     number_voxels_columns=attributes_training.shape[1]
162     #get the one of the dimensions of the cuboid grid
163     dimension_axis=int(round(math.pow(number_voxels_columns,1/3.))
164   ))
165     #transfrom the attributes
166     attributes_training=attributes_training.reshape(-1,
167     dimension_axis,\n
168       dimension_axis, dimension_axis,1)
169     #testing attributes
170     attributes_testing=attributes_testing.reshape(-1,
171     dimension_axis,\n
172       dimension_axis, dimension_axis,1)
173     #One hot encode training and testing labels
174     labels_training=labels=keras.utils.to_categorical(
175     labels_training)
176
177     #Generate the callbacks for saving the model
178     saving=IID_classifier.generate_model_saving_criteria\
179     (monitor_metric='val_accuracy',model_name=\
180     directory_results+'/'+'best_model'+\
181     ','+title_data_analysis+' '+str(analysis_index))
182
183     #Generate callbacks to stop the training porcess. patience
184     #is the number of iterations without an improvements
185     stopping=IID_classifier.generate_stopping_criteria\
186     (monitor_metric='val_accuracy',callback_patience=patience)
187     #generate a list with the callbacks functions we just created
188     list_callbacks=[saving,stopping]
189
190     #Traning process
191     IID_model_I_train= IID_classifier.\
192     train_model(IID_model_I, attributes_training,\n
193     labels_training, list_callbacks)
194
195     #Calculate the accuracy and plot the evalution of the
196     #training and validation performance throughout the
197     #training process for both models.
198     IID_model_I_accuracy=\
199     IID_classifier.model_evaluation\
200     (IID_model_I,attributes_testing,labels_testing)
201
202     #get the max validation score during the training process
203     max_val_score=max(IID_model_I_train.history['val_accuracy'])
204
205     #append the max validation score to the array
206     results_model_array[analysis_index]=max_val_score
```

```
204         #plot the evolution of the validation and training score
205         IIID_classifier.plot_validation_score(IIID_model_I_train,\n206             path=directory_results+'/'+voxnet+'_'+title_data_analysis+',\n207             +\\
208             str(analysis_index), title='voxnet '+title_data_analysis+',\n209             +\\
210             str(analysis_index))\n211\n212         #Get the model accuracy and plot the confusion matrix on the\n213         #training set
214         IIID_model_I_accuracy=\n215         IIID_classifier.model_evaluation(IIID_model_I,\n216             attributes_testing,\n217             labels_testing, path=\n218                 directory_results+'/'+confusion_matrix_voxnet +'+'+\\
219                 title_data_analysis+', '+str(analysis_index), title=\n220                 'confusion matrix voxnet '+title_data_analysis+', '+str(\n221                 analysis_index))\n222\n223         #Store the model accuracy
224         results_model_accuracy_array[analysis_index]=
225             IIID_model_I_accuracy\n226\n226         del IIID_model_I
227         del IIID_model_I_train
228         del IIID_model_I_accuracy
229         del list_callbacks\n230\n231         #calculate the mean and std of the obtained results
232         mean_val_accuracy=round(np.mean(results_model_array),3)
233         std_val_accuracy=round(np.std(results_model_array),3)
234         mean_accuracy=round(np.mean(results_model_accuracy_array),3)
235         std_accuracy=round(np.std(results_model_accuracy_array),3)\n236\n237         #Move the results to the results array
238         results_array[0,dataset_index]=mean_val_accuracy
239         results_array[1,dataset_index]=std_val_accuracy
240         results_array[2,dataset_index]=mean_accuracy
241         results_array[3,dataset_index]=std_accuracy\n242\n243         #transform the numpy array with the results into a dataframe
244         dataframe_results=pd.DataFrame(data=results_array,index=[mean
245             validation',\n246             'std validation','mean accuracy','std accuracy'],columns=
247             list_datasets)\n248         #transform the data frame into csv
249         export_csv=dataframe_results.to_csv(\n250             directory_results+'/'+csv_title+'.csv', index=True, header=True)\n251\n252 #######\n253\n254 #
```

```

#####
248 #Create the first and second models
249 list_data_sets=['merged_dataset_0.2labelled_instances.h5', \
250 'merged_dataset_0.4labelled_instances.h5', \
251 'merged_dataset_0.6labelled_instances.h5', \
252 'merged_dataset_0.8labelled_instances.h5','merged_dataset.h5']
253
254 titles=['0.20 dataset','0.40 dataset','0.60 dataset','0.80 dataset', \
255 'full dataset',]
256
257 model_names=['voxnet']
258
259
260 multiple_data_model_analysis(list_data_sets,titles,model_names, \
261 number_analysis_per_dataset=5,patience=30, csv_title='csv_multiple_data')

```

Listing 3: Multiple 3D data analysis

```

1 #Import standard libraries
2 import os
3 import numpy as np
4 import h5py
5 import matplotlib.pyplot as plt
6 import open3d as o3d
7 from mpl_toolkits.mplot3d import Axes3D
8 import numpy as np
9 import matplotlib.pyplot as plt
10 plt.style.use('seaborn-white')
11 import scipy.io as io
12 import scipy.ndimage as nd
13 import random
14
15 #functions we are using
16 #function to manage directories
17 def half_voxels_dimension(voxel_array_to_transform):
18     number_voxels=voxel_array_to_transform.shape[0]
19     #Get the number of voxels per dimension
20     one_dimension_voxels=voxel_array_to_transform.shape[1]
21     half_dimension=int(one_dimension_voxels/2)
22
23     #create an array to store the transformed instances
24     voxel_array_transformed=np.zeros((number_voxels,half_dimension, \
25     half_dimension,half_dimension,1))
26     #loop through all the instances in the array we want to transform
27     for voxel_transformation_index in range(number_voxels):
28         #get the array that we want to transform
29         voxel_to_transform=voxel_array_to_transform[ \
30         voxel_transformation_index]
31         #modify the voxels
32         voxel_to_transform=np.pad(voxel_to_transform,(0, 0), \
33         'constant',constant_values=(0, 0))
34         #Square voxels
            voxel_to_transform=nd.zoom(voxel_to_transform, \

```

```
35     (0.5, 0.5, 0.5), mode='constant', order=0)\\
36     .reshape((half_dimension,half_dimension,half_dimension,1))
37     #Add the transformed voxel to the array that contains the
38     #transformed
39     #voxels
40     voxel_array_transformed[voxel_tranformation_index]=\
41     voxel_to_transform
42     #return the modified array
43     return voxel_array_transformed
44
45 def double_voxels_dimension(voxel_array_to_transform):
46     number_voxels=voxel_array_to_transform.shape[0]
47     #Get the number of voxels per dimension
48     one_dimension_voxels=voxel_array_to_transform.shape[1]
49     double_dimension=one_dimension_voxels*2
50     print (double_dimension)
51     #create an array to store the transformed instances
52     voxel_array_transformed=np.zeros((number_voxels,double_dimension,\n
53     double_dimension,double_dimension,1))
54
55     #loop through all the instances in the array we want to transfrom
56     for voxel_tranformation_index in range(number_voxels):
57         #get the array that we want to transform
58         voxel_to_transform=voxel_array_to_transform[
59             voxel_tranformation_index]
60         #modify the voxels
61         voxel_to_transform=np.pad(voxel_to_transform,(0, 0),\
62             'constant',constant_values=(0, 0))
63         #Square voxels
64         voxel_to_transform=nd.zoom(voxel_to_transform,\
65             (2, 2, 2), mode='constant', order=0)\\
66             .reshape((double_dimension,double_dimension,double_dimension,1))
67         #Add the transformed voxel to the array that contains the
68         #transformed
69         #voxels
70         voxel_array_transformed[voxel_tranformation_index]=\
71         voxel_to_transform
72     #return the modified array
73     return voxel_array_transformed
74
75 def get_list_elements_pattern_not_current_directory(directory_to_search,
76 pattern):
77     #Comprehension list that by given a directory, explores
78     pattern_files= [element for element in os.listdir(directory_to_search)
79     ) if \
80     element.endswith("." +pattern)]
81     #
82     return pattern_files
83
84 def get_list_elements_without_pattern_not_current_directory(
85 directory_to_search):
86     #Comprehension list that by given a directory, explores
```

```
82     files= [element for element in os.listdir(directory_to_search)]
83     #
84     return files
85
86 #function to get automatically the path of a given file in the current
87 #directory
88 def generate_directory_path_contains_current_directory(folder_name):
89     #Get current directory
90     current_directory=os.getcwd()
91     #Create a directory path to explore
92     directory_to_explore=current_directory+'/'++folder_name
93     #Create a directory path to explore
94     return directory_to_explore
95
96 def create_folder_in_path_check_folder_created(path_creation,
97                                               path_to_create):
98     #
99     directories_in_directory_where_eant_create=\
100    get_list_elements_without_pattern_not_current_directory(path_creation)
101    #
102    directories_path_in_directory_where_eant_create=\
103    [path_creation+'/'++path for path in \
104    directories_in_directory_where_eant_create]
105    #
106    if path_to_create not in
107    directories_path_in_directory_where_eant_create:
108        #
109        os.mkdir(path_to_create)
110
111 #function to plot the loss functions
112 def loss_gans_plot(evolution_loss_function_discriminator,\n113 evolution_loss_function_generator, evolution_accuracy, results_directory,\n114 label):\n115     plt.figure(figsize=(20,20), dpi=80)\n116\n117     #First subgraph with discriminator loss\n118     plt.rc('xtick',labelsize=30)\n119     plt.rc('ytick',labelsize=30)\n120     plt.subplot(3, 1, 1)\n121     plt.plot(evolution_loss_function_discriminator,'darkorange', lw=0.4)\n122     plt.yscale('log')\n123     plt.title("Evolution of Discriminator's Loss", fontsize=30)\n124     plt.xlabel('epochs',fontsize=30)\n125     plt.ylabel("Discriminator's Loss (log scale)",fontsize=30)\n126\n127     #Second subgraph with generator loss\n128     plt.subplot(3, 1, 2)\n129     plt.plot(evolution_loss_function_generator,'blue', lw=0.4)\n130     plt.title("Evolution of Generator's Loss", fontsize=30)\n131     plt.rc('xtick',labelsize=30)\n132     plt.rc('ytick',labelsize=30)
```

```
130     plt.xlabel('epochs', fontsize=30)
131     plt.ylabel("Generator's Loss", fontsize=30)
132
133
134     #Third subgraph with accuracy
135
136     plt.subplot(3, 1, 3)
137     plt.plot(evolution_accuracy, 'forestgreen', lw=0.45)
138     plt.title("Evolution of Discriminator's Accuracy", fontsize=30)
139     plt.rc('xtick', labelsize=30)
140     plt.rc('ytick', labelsize=30)
141     plt.xlabel('epochs', fontsize=30)
142     plt.ylabel("Discriminator's Accuracy", fontsize=30)
143     #plt.plot(evolution_loss_function_generator, lw=1)
144
145     #Space between subgraphs in the main graph
146     plt.subplots_adjust(hspace=0.4)
147
148     #plt.show()
149     plt.savefig(results_directory+ '/_evolution_loss_accuracy'+str(label))
150
151 def get_loss_data_from_checkpoints(data_directory, checkpoint_to_append):
152     evolution_loss_function_discriminator=[]
153     evolution_loss_function_generator=[]
154     evolution_accuracy=[]
155     elements_to_explore=['discriminator_loss','generator_loss',\
156     'discriminator_accuracy']
157     number_elements_to_explore=len(elements_to_explore)
158
159     for elements_to_explore_index in range(number_elements_to_explore):
160         element_to_explore=elements_to_explore[elements_to_explore_index]
161
162         for checkpoint_index_dis in range(len(checkpoint_to_append)):
163             checkpoint=checkpoint_to_append[checkpoint_index_dis]
164             #get the dataset
165             data_path=data_directory+ '/evolution_loss_functions' +
166             checkpoint+'.h5'
167
168             dataset= h5py.File(data_path, 'r')
169             loss_values=np.array(dataset.get(element_to_explore))
170             dataset.close()
171             del dataset
172
173             for information in loss_values:
174                 if elements_to_explore_index==0:
175                     evolution_loss_function_discriminator.append(
176                         information)
177                 elif elements_to_explore_index==1:
178                     evolution_loss_function_generator.append(information)
179                 elif elements_to_explore_index==2:
180                     evolution_accuracy.append(information)
181
182     return evolution_loss_function_discriminator,
```

```
    evolution_loss_function_generator, evolution_accuracy
181
182 def display_generated_data(data_path, list_epochs, \
183 epoch_in_list_to_display=0, instance_to_display=0, binarise=True, \
184 reduce_voxel=False):
185     #get the path of the dataset we want
186     epoch_to_analyse=list_epochs[epoch_in_list_to_display]
187
188     #get the directory of the epoch
189     generated_data_path=data_path+ '/generated_data_array_'+
190     epoch_to_analyse+'.h5'
191
192     #Get the dataset
193     dataset= h5py.File(generated_data_path, 'r')
194     generated_data=np.array(dataset.get('generated_data'))
195     dataset.close()
196
197     #get the instance of the set we want to display
198     generated_instance=generated_data[instance_to_display]
199
200     if binarise == True:
201         generated_instance=np.where(generated_instance>=0.5,1,0)
202
203     if reduce_voxel==True:
204         generated_instance=nd.zoom(generated_instance, \
205             (0.5, 0.5, 0.5), mode='constant', order=0)\

206     #information about the reshape of the instance to dispalnce and
207     #reshaping
208     generated_instance_shape_voxel=generated_instance.shape[0]
209
210     generated_instance=generated_instance.reshape((\
211         generated_instance_shape_voxel,generated_instance_shape_voxel, \
212         generated_instance_shape_voxel))
213
214     #plot
215     fig=plt.figure()
216     ax = fig.gca(projection='3d')
217     ax.grid(False)
218     plt.axis('off')
219     ax.voxels(generated_instance, facecolors='aqua', edgecolor="k")
220     plt.show()
221
222 def original_data(data_path, list_epochs, \
223 label=0, instance_to_display=0, binarise=True, reduce_voxel=False):
224     #get the directory of the epoch
225     generated_data_path=data_path
226
227     #Get the dataset
228     dataset= h5py.File(generated_data_path, 'r')
229     generated_data=np.array(dataset.get('attributes_training'))
     labels_training=np.array(dataset.get('labels_training'))
```

```
230     dataset.close()
231
232     #filter by label
233     filter=np.where(labels_training==label)
234     generated_data=generated_data[filter]
235     labels_training=labels_training[filter]
236
237     #get the instance of the set we want to display
238     generated_instance=generated_data[instance_to_display]
239
240     generated_instance=generated_instance.reshape((32,32,32))
241
242     if binarise == True:
243         generated_instance=np.where(generated_instance>=0.5,1,0)
244
245     if reduce_voxel==True:
246         generated_instance=nd.zoom(generated_instance,\n247             (0.5, 0.5, 0.5), mode='constant', order=0)\n248
249
250     #information about the reshape of the instance to dispalnce and\n251     #reshaping
252
253     generated_instance=np.pad(generated_instance,(0, 0),\n254         'constant',constant_values=(0, 0))
255     #Square voxels
256     generated_instance=nd.zoom(generated_instance,\n257         (2, 2, 2), mode='constant', order=0)\n258     .reshape((64,64,64,1))
259
260     generated_instance_shape_voxel=generated_instance.shape[0]
261
262     generated_instance=generated_instance.reshape(\n263         generated_instance_shape_voxel,generated_instance_shape_voxel,\n264         generated_instance_shape_voxel)
265
266     #plot
267     fig=plt.figure()
268     ax = fig.gca(projection='3d')
269     ax.grid(False)
270     plt.axis('off')
271     ax.voxels(generated_instance,facecolors='aqua', edgecolor="k")
272     plt.show()
273
274 def get_path_generated_data(data_path,list_epochs,\n275 epoch_in_list_to_display=0,instance_to_display=0,binarise=True,\n276     reduce_voxel=False):
277     #get the path of the dataset we want
278     epoch_to_analyse=list_epochs[epoch_in_list_to_display]
279
280     #get the directory of the epoch
281     generated_data_path=data_path+='/generated_data_array_+'\n282     epoch_to_analyse+'.h5'
```

```
280     return generated_data_path
281
282 #function to merge dataset. To original datasets and the generated one
283 def merge_dataset_with_augmented(augmented_path,
284     path_dataset_to_be_augmented,\n
285     filters=[1,2,3], filter=True, label=2, reduce_voxels=True):
286
287     #get the dataset we want to increase and the subset within it
288     dataset= h5py.File(path_dataset_to_be_augmented, 'r')
289     attributes_training=np.array(dataset.get('attributes_training'))
290     attributes_testing=np.array(dataset.get('attributes_testing'))
291     labels_training=np.array(dataset.get('labels_training'))
292     labels_testing=np.array(dataset.get('labels_testing'))
293     dataset.close()
294
295     #Loop through all the augmented dataset and add the to the original
296     #set
297     #including the labels
298
299     augmented_dataset= h5py.File(augmented_path, 'r')
300     generated_data=np.array(augmented_dataset.get("generated_data"))
301     labels_generated_data=np.array(augmented_dataset.get("labels"))
302
303     print (generated_data.shape)
304
305     if filter == True:
306         generated_data=generated_data[filters]
307         number_instances=generated_data.shape[0]
308         label_array=np.full((number_instances),label)
309         print (label_array)
310         labels_generated_data=label_array
311         print (labels_generated_data)
312
313     generated_data=generated_data.reshape((generated_data.shape[0],
314     generated_data.shape[1],generated_data.shape[2],generated_data.shape
315     [3]))
316
317     #Reduce the size of the generated data
318     if reduce_voxels==True:
319         generated_data=half_voxels_dimension(generated_data)
320
321     generated_data=generated_data.reshape((generated_data.shape[0],32768)
322     )
323
324     print (attributes_training.shape)
325     print (generated_data.shape)
326     #concatenate the data sets
327     attributes_training=np.concatenate((attributes_training,
328     generated_data),axis=0)
329     labels_training=np.concatenate((labels_training,labels_generated_data
330     ),axis=0)
331
332     #safety prints
333     print(generated_data.shape)
```

```
326     print (labels_generated_data.shape)
327     print (attributes_training.shape)
328     print (labels_training.shape)
329     unique, counts = np.unique(labels_training, return_counts=True)
330     print (unique)
331     print (counts)
332
333     #save the dataset we have augmented
334
335     with h5py.File(path_dataset_to_be_augmented, 'w') as hf:
336         hf.create_dataset('attributes_training', data=attributes_training)
337         hf.create_dataset('attributes_testing', data=attributes_testing)
338         hf.create_dataset('labels_training', data=labels_training)
339         hf.create_dataset('labels_testing', data=labels_testing)
340     hf.close()
341
342 def merge_dataset_with_original(original_dataset_path,
343                                 path_dataset_to_be_augmented,\n343                                 label=0, filters=[1,2,3], filter=True, reduce_voxels=True):
344
345     #get the dataset we want to increase and the subset within it
346     dataset= h5py.File(path_dataset_to_be_augmented, 'r')
347     attributes_training=np.array(dataset.get('attributes_training'))
348     attributes_testing=np.array(dataset.get('attributes_testing'))
349     labels_training=np.array(dataset.get('labels_training'))
350     labels_testing=np.array(dataset.get('labels_testing'))
351     dataset.close()
352
353     #get the augmented data of a specific label
354     augmented_dataset= h5py.File(original_dataset_path, 'r')
355     generated_data=np.array(augmented_dataset.get('attributes_training'))
356     labels_generated_data=np.array(augmented_dataset.get('labels_training'))
357     augmented_dataset.close()
358
359     filter_I=np.where(labels_generated_data==label)
360     generated_data=generated_data[filter_I]
361     labels_generated_data=labels_generated_data[filter_I]
362
363     generated_data=generated_data(filters]
364     labels_generated_data=labels_generated_data[filters]
365
366     #concatenate the data sets
367     attributes_training=np.concatenate((attributes_training,
368                                         generated_data),axis=0)
369     labels_training=np.concatenate((labels_training,labels_generated_data),
368                                         axis=0)
370
371     #safety prints
372     print(generated_data.shape)
373     print (labels_generated_data.shape)
374     print (attributes_training.shape)
375     print (labels_training.shape)
```

```
375     unique, counts = np.unique(labels_training, return_counts=True)
376     print (unique)
377     print (counts)
378
379     #save the dataset we have augmented
380     with h5py.File(path_dataset_to_be_augmented, 'w') as hf:
381         hf.create_dataset('attributes_training', data=attributes_training)
382         hf.create_dataset('attributes_testing', data=attributes_testing)
383         hf.create_dataset('labels_training', data=labels_training)
384         hf.create_dataset('labels_testing', data=labels_testing)
385     hf.close()
386
387
388 ##### END OF FUNCTIONS #####
389
390
391 #
392 ##### PATHS/DIRECTORIES #####
393
394 #visualization tool for generated data and for the evalution of the loss
395 #functions
396
397 #Set up directories
398 #root directory
399 current_directory=os.getcwd()
400
401 #directory where the data is stored
402 data_directory='G:/gans_project_root_directory/processed_data/
   gans_results_0.2/7/checkpoints_and_arrays'
403
404 #from the data directory get the name of our data
405 data_names=\
406 get_list_elements_pattern_not_current_directory(data_directory,'h5')
407
408 #set up directory where we are going to save the visualisations
409 visualizations_directory='G:/gans_project_root_directory/visualizations/
   visualization_IID_figures/generation_visualization/2'
410
411 if not os.path.exists(visualizations_directory):
412     os.makedirs(visualizations_directory)
413
414 loss_visualizations_directory='G:/gans_project_root_directory/
   visualizations/loss_functions/lr01'
415 if not os.path.exists(loss_visualizations_directory):
416     os.makedirs(loss_visualizations_directory)
417
418 new_generations_directory='G:/gans_project_root_directory/processed_data/
   gans_results/0_0.9_lr'
419 if not os.path.exists(new_generations_directory):
```

```

420     os.makedirs(new_generations_directory)
421
422 path_dataset_to_merge='G:/gans_project_root_directory/processed_data/
    voxel_datasets'
423 #
# ##########
424 #open the dataset and get the information we want
425 #####
426
427
428 #epochs to get from a data set
429 epoch_to_get=['5000']
430
431 ##### VISUALISATION GENERATED #####
432 display=True
433 if display ==True:
434     display_generated_data(data_directory,epoch_to_get,\n
        epoch_in_list_to_display=0,instance_to_display=40, binarise=True,\n
        reduce_voxel=False)
435
436 ##### VISUALISATION ORIGINAL #####
437 #visualise the original data and not the augmented
438 display_original=False
439 original_set_path='G:/gans_project_root_directory/processed_data/
    voxel_datasets/merged_dataset.h5'
440 if display_original==True:
441     original_data(original_set_path,epoch_to_get,\n
        label=4,instance_to_display=68,binarise=True,reduce_voxel=False)
442
443 #Because lack of memory RAM the process stops several times, hence I have
    to
444 #merge the results of several points where the process stopped
445
446 #checkpoint label 0
447 dictionary_checkpoint_to_get={ '0': ['1050', '2750', '4650', '5500'],
448 '1': ['1350', '3050', '3950', '4550', '5150'], '2': ['1800', '4250', '5950'], \
449 '3': ['350', '3850', '4550', '5500'], '4': ['550', '2100', '5950'], \
450 '5': ['1600', '2350', '4150', '5200', '5950'], '6': ['800', '5900'], \
451 '7': ['650', '3850', '4950', '5850', '5950', '6450'], \
452 '8': [], '9': [], '10': [], '11': [], '12': [], '13': [], \
453 'lr01': ['5650', '4450', '4250', '4050', '3900', '3800', '2950', '2800', '2650', \
        '2500', \
454 '2350', '1600', '1450', '1300', '1150', '1000', '800', '600', '450', '250'], \
455 'lr01': ['5850', '3700', '3550', '3350', '2350', '2250', '600', '450', '250']}
456
457 ##### CHECK CHECKPOINT GANS #####
458 experimenting_checkpoint=False
459 if experimenting_checkpoint==True:
460     path_to_open=data_directory+ '/evolution_loss_functions250.h5'

```

```
463
464     dataset= h5py.File(path_to_open , 'r')
465     loss_values=np.array(dataset.get('dicriminator_loss'))
466     dataset.close()
467     del dataset
468     print (len(loss_values))
469
470
471 ##### VISUALISATION LOSS
472 #####
473 visualise=False
474 if visualise == True:
475     evolution_loss_function_discriminator ,
476     evolution_loss_function_generator ,\
477     evolution_accuracy=\
478     get_loss_data_from_checkpoints(data_directory ,
479     dictionary_checkpoint_to_get['lr01'])
480
481
482
483
484 ##### ANALYSIS AUGMENTED SET
485 #####
486 #Analysis of the dataset to be augmented for sanity check
487 analysis_set=False
488 if analysis_set==True:
489     #get the dataset we want to increase and the subset within it
490     dataset= h5py.File('G:/gans_project_root_directory/processed_data/
491     augmented_voxels_dataset_voxnet/0.2_augmented/augmented_0.2_10.h5' , 'r
492     ')
493     attributes_training=np.array(dataset.get('attributes_training'))
494     attributes_testing=np.array(dataset.get('attributes_testing'))
495     labels_training=np.array(dataset.get('labels_training'))
496     labels_testing=np.array(dataset.get('labels_testing'))
497     dataset.close()
498
499     uniqueValues , occurCount = np.unique(labels_training , return_counts=
500     True)
501     print (attributes_training.shape)
502     print (uniqueValues)
503     print (occurCount)
504     #np.savetxt(
505     #'G:/gans_project_root_directory/processed_data/
506     augmented_voxels_dataset_voxnet/0.2_augmented/0.2_shape.txt' ,\
507     #np.concatenate((uniqueValues , occurCount) , axis=0) , fmt="%s")
508
509 #
510 #####
511
512 list_generated_sets_paths=[]
```

```
506 #dat
507 #list of dataset to augment
508 datasets_to_augment=['G:/gans_project_root_directory/processed_data/
509     augmented_voxels_dataset_voxnet/full_augmented/augmented_full_10.h5', \
510 'G:/gans_project_root_directory/processed_data/
511     augmented_voxels_dataset_voxnet/full_augmented/augmented_full_20.h5', \
512 'G:/gans_project_root_directory/processed_data/
513     augmented_voxels_dataset_voxnet/full_augmented/augmented_full_30.h5', \
514 'G:/gans_project_root_directory/processed_data/
515     augmented_voxels_dataset_voxnet/full_augmented/augmented_full_40.h5', \
516 'G:/gans_project_root_directory/processed_data/
517     augmented_voxels_dataset_voxnet/full_augmented/augmented_full_50.h5']
518
519 #get dataset to augment
520 augmentation=False
521 if augmentation==True:
522     for augmentation_set in [0,1,2,3,4]:
523         path_dataset_to_be_augmented=datasets_to_augment[augmentation_set]
524     ]
525         print (path_dataset_to_be_augmented)
526         analysis_set=True
527         if analysis_set==True:
528             #get the dataset we want to increase and the subset within it
529             dataset= h5py.File(path_dataset_to_be_augmented, 'r')
530             attributes_training=np.array(dataset.get('attributes_training'))
531         attributes_testing=np.array(dataset.get('attributes_testing'))
532     )
533         labels_training=np.array(dataset.get('labels_training'))
534         labels_testing=np.array(dataset.get('labels_testing'))
535         dataset.close()
536         uniqueValues, occurCount = np.unique(labels_training,
537         return_counts=True)
538         print (attributes_training.shape)
539         print (uniqueValues)
540         print (occurCount)
541
542         if augmentation_set == 0:
543             random_generation=3
544
545         if augmentation_set == 1:
546             random_generation=7
547
548         if augmentation_set == 2:
549             random_generation=10
550
551         if augmentation_set == 3:
552             random_generation=13
553
554         if augmentation_set == 4:
555             random_generation=18
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
274
```

```
550     filter_random=random.sample(range(1, 40), random_generation)
551     #results path
552     results_path='G:/gans_project_root_directory/processed_data/
553     augmented_voxels_dataset_voxnet'
554
554     augmented_dataset_name_list=['augmented_0.20_dataset',
555     'augmented_0.40_dataset',
556     'augmented_0.60_dataset', 'augmented_0.80_dataset', 'augmented_full
557     dataset']
558
558     ##### AUGMENTATION
559     #####
560     generated_dataset_path='G:/gans_project_root_directory/
561     processed_data/gans_results/13/checkpoints_and_arrays/
562     generated_data_array_5250.h5'
563
563     #augmenting the data
564     augmentation=True
565     filter_random=random.sample(range(1, 40), random_generation)
566     if augmentation==True:
567         merge_dataset_with_augmented(generated_dataset_path,
568         path_dataset_to_be_augmented,\n
569             filters=filter_random, filter=True, label=13, reduce_voxels=
570             True)
571
571     generated_dataset_path='G:/gans_project_root_directory/
572     processed_data/gans_results/13/checkpoints_and_arrays/
573     generated_data_array_5250.h5'
574
574     #augmenting the data
575     augmentation=True
576     filter_random=random.sample(range(1, 40), random_generation)
577     if augmentation==True:
578         merge_dataset_with_augmented(generated_dataset_path,
579         path_dataset_to_be_augmented,\n
580             filters=filter_random, filter=True, label=13, reduce_voxels=
581             True)
582
582 #
583 #####
584
```

```

585 augmentation_original=False
586 if augmentation_original==True:
587     path_dataset_to_be_augmented=datasets_to_augment[4]
588     print (path_dataset_to_be_augmented)
589     ##### To delete #####
590     original_set_path='G:/gans_project_root_directory/processed_data/
591 voxel_datasets/merged_dataset.h5'
592     for label_loop in [0,1,2,3,4,5,6,7,8,9,10,11,12,13]:
593         filters_random=filter_random=random.sample(range(1, 40), 5)
594         merge_dataset_with_original(original_set_path,
595         path_dataset_to_be_augmented,\n
596         label=label_loop, filters=filters_random, filter=True,
597         reduce_voxels=False)
598         for label_loop in [11,12]:
599             filters_random=filter_random=random.sample(range(1, 40), 2)
600             merge_dataset_with_original(original_set_path,
601             path_dataset_to_be_augmented,\n
602             label=label_loop, filters=filters_random, filter=True,
603             reduce_voxels=False)
604
605 analysis_set=False
606 if analysis_set==True:
607     #get the dataset we want to increase and the subset within it
608     dataset= h5py.File(path_dataset_to_be_augmented, 'r')
609     attributes_training=np.array(dataset.get('attributes_training'))
610     attributes_testing=np.array(dataset.get('attributes_testing'))
611     labels_training=np.array(dataset.get('labels_training'))
612     labels_testing=np.array(dataset.get('labels_testing'))
613     dataset.close()
614
615     uniqueValues, occurCount = np.unique(labels_training, return_counts=
616     True)
617     print (attributes_training.shape)
618     print (uniqueValues)
619     print (occurCount)
620     #np.savetxt(\
```

Listing 4: Augmentation and visualisation

```

1 #Import standard libraries
2 import os
3 import numpy as np
4 import pandas as pd
5 import sys
6 import csv
7 import requests
8 import xml.etree.ElementTree as ET
9 import xmltodict
10 import shutil
11 import h5py
12 import open3d as o3d
13 from mpl_toolkits.mplot3d import Axes3D
```

```
14 import numpy as np
15 import matplotlib.pyplot as plt
16
17 #glob lists the elements in the current directory with a specific pattern
18 import glob
19 from matplotlib import pyplot as plt
20 import matplotlib.patches as patches
21 from voxelgrid import VoxelGrid
22 from mpl_toolkits.mplot3d import Axes3D
23 from sklearn import preprocessing
24
25
26 #functions we are using
27 #function to manage directories
28 def get_list_elements_pattern_not_current_directory(directory_to_search,
29   pattern):
30   #Comprehension list that by given a directory, explores
31   pattern_files= [element for element in os.listdir(directory_to_search)
32     ) if \
33       element.endswith("." +pattern)]
34   #
35   return pattern_files
36
37 def get_list_elements_without_pattern_not_current_directory(
38   directory_to_search):
39   #Comprehension list that by given a directory, explores
40   files= [element for element in os.listdir(directory_to_search)]
41   #
42   return files
43
44 #function to get automatically the path of a given file in the current
45 #directory
46 def generate_directory_path_contains_current_directory(folder_name):
47   #Get current directory
48   current_directory=os.getcwd()
49   #Create a directory path to explore
50   directory_to_explore=current_directory+'/' +folder_name
51   #Create a directory path to explore
52   return directory_to_explore
53
54 def create_folder_in_path_check_folder_created(path_creation,
55   path_to_create):
56   #
57   directories_in_directory_where_eant_create=\
58   get_list_elements_without_pattern_not_current_directory(path_creation
59   )
59   #
60   directories_path_in_directory_where_eant_create=\
61   [path_creation+'/'+path for path in \
62   directories_in_directory_where_eant_create]
63   #
64   if path_to_create not in
65   directories_path_in_directory_where_eant_create:
```

```
60     #
61     os.mkdir(path_to_create)
62
63 def open_obj_to_data_frame(obj_file):
64     data = pd.read_csv(obj_file, delimiter=' ', names=['cat', 'x', 'y', 'z'], \
65     skiprows=2)
66
67     data_frame_point_cloud=data.loc[data['cat'] == 'v']
68
69     return data_frame_point_cloud[['x', 'y', 'z']]
70
71 def normalize_dataframe_to_array(dataframe):
72     #Name of the coordinates. We use this to parse the dataframe
73     coordinates=['x', 'y', 'z']
74
75     #Shape of the dataframe. We use this info to create a numpy array with
76     #the same characteristics
77     number_points=dataframe.shape[0]
78     number_coordinates=dataframe.shape[1]
79
80     #Create a dataframe
81     normalised_pointcloud_array=np.zeros((number_points,
82     number_coordinates))
83
84     counter=0
85     #Loop through the coordinate
86     for coordinate in coordinates:
87         #Get the column we want to normalise
88         column_to_normalise=np.array(dataframe[coordinate].values.\
89         astype(float)).reshape(-1,1)
90
91         #Get the normalizer
92         min_max_scaler=preprocessing.MinMaxScaler()
93
94         #Normalise the column
95         normalised_column= min_max_scaler.fit_transform(
96         column_to_normalise)
97
98         #put the normalised column into the normalised array
99         normalised_pointcloud_array[:,counter]=normalised_column.flatten()
100
101         #Add one to the counter
102         counter=counter+1
103
104 #Function to voxelize a single point cloud using the functions in the
105 #voxelgrid
106 def cloud_voxelize_binary_values(poin_cloud, voxgrid_dimension=[32,32,32]):
107     :
108     #Get the voxel object
109     grid=VoxelGrid(poin_cloud, x_y_z=voxgrid_dimension)
```

```
108 #From the voxel object get an array that indicates the number of
109 point
110 #within each boxel
111 dimensional_cuadatric_array=np.array(grid.vector)
112
113 #if a voxel is not empty assign value 1 to the voxel. Otherwise,
114 #assign the
115 #value 0
116 dimensional_cuadatric_array=np.where(dimensional_cuadatric_array
117 >0,1,0)
118 #
119 number_voxels=voxgrid_dimension[0]*voxgrid_dimension[1]*\
120 voxgrid_dimension[2]
121 #
122 dimensional_cuadatric_array=\
123 dimensional_cuadatric_array.reshape(1,number_voxels)
124 #
125 return dimensional_cuadatric_array
126
127 #Set up directories
128 #root directory
129 current_directory=os.getcwd()
130 #directory where the data is stored
131 data_directory='G:/gans_project_root_directory/hips/50004_hips/'
132 #from the data directory get the name of our data
133
134 data_names=\
135 get_list_elements_pattern_not_current_directory(data_directory,'obj')
136
137 #set up directory where we are going to save the visualisations
138 visualizations_directory='G:/gans_project_root_directory/visualizations'
139 #create the visualization directory
140
141 create_folder_in_path_check_folder_created(current_directory, \
142 visualizations_directory)
143
144 for element_index in range(len(data_names)):
145     #get the obj that we want to
146     obj_item_path=data_directory+'/'+data_names[element_index]
147
148     #open the file and transform it to a normalise pointcloud
149     point_cloud=open_obj_to_data_frame(obj_item_path)
150     point_cloud=normalize_dataframe_to_array(point_cloud)
151
152     #od3 object and point cloud plotting
153     three_dimensional_object=o3d.geometry.PointCloud()
154     three_dimensional_object.points=o3d.utility.Vector3dVector(
155     point_cloud)
156     o3d.visualization.draw_geometries([three_dimensional_object])
157
158     #voxels 64
```

```

157     sixty_four_voxel=\n158     cloud_voxelize_binary_values(point_cloud,voxgrid_dimension\n159     =[64,64,64])\n160\n161     fig = plt.figure()\n162     ax = fig.gca(projection='3d')\n163     ax.grid(False)\n164     plt.axis('off')\n165     ax.voxels(sixty_four_voxel.reshape((64,64,64)),facecolors='aqua',\n166     edgecolor="k")\n167     plt.show()\n168\n169 #voxel 32\n170 three_two_voxel=\n171 cloud_voxelize_binary_values(point_cloud,voxgrid_dimension\n172     =[32,32,32])\n173\n174     fig = plt.figure()\n175     ax = fig.gca(projection='3d')\n176     ax.grid(False)\n177     plt.axis('off')\n178     ax.voxels(three_two_voxel.reshape((32,32,32)),facecolors='aqua',\n179     edgecolor="k")\n180     plt.show()

```

Listing 5: Visualisation of obj files and triangular meshes

```

1 #Import standard libraries\n2 import os\n3 import numpy as np\n4 import pandas as pd\n5 import sys\n6 import csv\n7 import requests\n8 import xml.etree.ElementTree as ET\n9 import xmltodict\n10 import shutil\n11 import h5py\n12\n13 #glob lists the elements in the current directory with a specific pattern\n14 import glob\n15 from matplotlib import pyplot as plt\n16 import matplotlib.patches as patches\n17 from voxelgrid import VoxelGrid\n18 from mpl_toolkits.mplot3d import Axes3D\n19 from sklearn import preprocessing\n20\n21 def get_list_elements_without_pattern_not_current_directory(\n22     directory_to_search):\n23     #Comprehension list that by given a directory, explores\n24     files= [element for element in os.listdir(directory_to_search)]\n25     #\n26     return

```

```
27 def get_list_elements_pattern_not_current_directory(directory_to_search,
28     pattern):
29     #Comprehension list that by given a directory, explores
30     pattern_files= [element for element in os.listdir(directory_to_search)
31     ) if \
32         element.endswith("." +pattern)]
33     #
34     return pattern_files
35
36
37
38
39
40 #function to get automatically the path of a given file in the current
41 #directory
42 def generate_directory_path_contains_current_directory(folder_name):
43     #Get current directoy
44     current_directory=os.getcwd()
45     #Create a directory path to explore
46     directory_to_explore=current_directory+ '/' +folder_name
47     #Create a directory path to explore
48     return directory_to_explore
49
50
51
52
53
54
55
56
57
58
59
60
61
62 #fucntion to create folders in a given path
63 def create_folders_in_path(path, folder_names_list):
64     #create folders in a given path. The folders names are given by a
65     #list
66     for folder_name in folder_names_list:
67         path_new_directory= path+ '/' + folder_name
68         create_folder_in_path_check_folder_created(path,
69         path_new_directory)
70 ##### OBJ TRANSFORMATION
```

```
#####
71 #
72 def open_obj_to_data_frame(obj_file):
73     data = pd.read_csv(obj_file, delimiter=' ', names=['cat', 'x', 'y', 'z'], \
74     skiprows=2)
75
76     data_frame_point_cloud=data.loc[data['cat'] == 'v']
77
78     return data_frame_point_cloud[['x', 'y', 'z']]
79
80 #
81 def normalize_dataframe_to_array(dataframe):
82     #Name of the coordinates. We use this to parse the dataframe
83     coordinates=['x', 'y', 'z']
84
85     #Shape of the dataframe. We use this info to create a numpy array with
86     #the same characteristics
87     number_points=dataframe.shape[0]
88     number_coordinates=dataframe.shape[1]
89
90     #Create a dataframe
91     normalised_pointcloud_array=np.zeros((number_points,
92                                         number_coordinates))
93
94     counter=0
95     #Loop through the coordinate
96     for coordinate in coordinates:
97         #Get the column we want to normalise
98         column_to_normalise=np.array(dataframe[coordinate].values.\
99             astype(float)).reshape(-1,1)
100
101         #Get the normalizer
102         min_max_scaler=preprocessing.MinMaxScaler()
103
104         #Normalise the column
105         normalised_column= min_max_scaler.fit_transform(
106             column_to_normalise)
107
108         #put the normalised column into the normalised array
109         normalised_pointcloud_array[:,counter]=normalised_column.flatten()
110
111         #Add one to the counter
112         counter=counter+1
113
114
115 #Function to voxelize a single point cloud using the functions in the
116 #voxelgrid
117 def cloud_voxelize_binary_values(poin_cloud,voxgrid_dimension=[32,32,32]):
118     :
119     #Get the voxel object
```

```
118     grid=VoxelGrid(poin_cloud, x_y_z=voxgrid_dimension)
119
120     #From the voxel object get an array that indicates the number of
121     #point
122     #within each boxel
123     dimensional_cuadatric_array=np.array(grid.vector)
124
125     #if a voxel is not empty assign value 1 to the voxel. Otherwise,
126     #assign the
127     #value 0
128     dimensional_cuadatric_array=np.where(dimensional_cuadatric_array
129     >0,1,0)
130     #
131     number_voxels=voxgrid_dimension[0]*voxgrid_dimension[1]*\
132     voxgrid_dimension[2]
133     #
134     dimensional_cuadatric_array=\n
135     dimensional_cuadatric_array.reshape(1,number_voxels)
136     #
137     return dimensional_cuadatric_array
138
139 #
140 def create_array_labels(label,number_instances):
141     #
142     label_array=np.full((number_instances),label)
143     #
144     return label_array
145
146 #
147 def transform_cloud_points_into_single_file_voxels(
148     path_contains_folders_we_want_analyse,\n
149     list_folders_to_parse,labels_list,voxgrid_size=[32,32,32]):
150     #
151     number_folders_explore=len(list_folders_to_parse)
152     #get the path of the folders that we want to explore given a path and
153     #the
154     #name of the folders
155     paths_to_explore=[path_contains_folders_we_want_analyse+'/'\
156     'results'+'/' +folder_to_parse for folder_to_parse in
157     list_folders_to_parse]
158
159     #
160     results_path=path_contains_folders_we_want_analyse+'/'+'\
161     results_voxels'
162     #
163     create_folder_in_path_check_folder_created(\n
164     path_contains_folders_we_want_analyse,results_path)
165
166     #create the paths of the folder to store the voxels
167     results_point_cloud_directories=[\n
168     path_contains_folders_we_want_analyse+'/'+'\
169     'results_voxels'+'/' +folder_to_analyse+'_'++'voxels' for
170     folder_to_analyse in\n
```

```
162     list_folders_to_parse]
163
164     #create the folders to store the results
165     for directory in results_point_cloud_directories:
166         create_folder_in_path_check_folder_created(\n
167             results_path,directory)
168
169     #
170     for folder_to_explore_index in range(number_folders_explore):
171         #
172         store_results_folder=results_point_cloud_directories\
173         [folder_to_explore_index]
174         #
175         folder_to_explore=paths_to_explore[folder_to_explore_index]
176         #
177         label=labels_list[folder_to_explore_index]
178         #
179         action_of_analysis=list_folders_to_parse[folder_to_explore_index]
180         #
181         elements_in_folder_to_explore=\n
182             get_list_elements_pattern_not_current_directory(folder_to_explore
183             ,\
184             'obj')
185         #
186         number_intems_to_voxelise=len(elements_in_folder_to_explore)
187         #
188         number_voxels=voxgrid_size[0]*voxgrid_size[1]*voxgrid_size[2]
189         #
190         voxel_matrix=np.zeros((number_intems_to_voxelise,number_voxels))
191         #
192         labels_array=create_array_labels(label,number_intems_to_voxelise)
193         #
194         for element_to_voxelize_index in range(number_intems_to_voxelise):
195             :
196             element_to_voxelise=folder_to_explore+ '/' + \
197                 elements_in_folder_to_explore[element_to_voxelize_index]
198             #
199             point_cloud=open_obj_to_data_frame(element_to_voxelise)
200             #
201             point_cloud=normalize_dataframe_to_array(point_cloud)
202             #
203             voxel_transformation=\n
204                 cloud_voxelize_binary_values(point_cloud,voxgrid_dimension=
205                     voxgrid_size)
206             #
207             voxel_matrix[element_to_voxelize_index]=voxel_transformation
208             #
209             with h5py.File(store_results_folder+'/'+action_of_analysis+'.h5'\
210             , 'w') as hf:
211                 hf.create_dataset('attributes', data=voxel_matrix)
212                 hf.create_dataset('labels', data=labels_array)
213             hf.close()
```

```

212     return 'done'
213
214
215 ##### IMPLEMENTATION
216 #####
217 #Folder to explore
218 folders_to_explore=['punching','running_on_spot','chicken_wings','hips',\
219 'knees','jumping_jacks','shake_arms','shake_shoulders','shake_hips',\
220 'one_leg_loose','one_leg_jump','light_hopping_loose','light_hopping_stiff
',\
221 'jiggle_on_toes']
222 #Jump list
223 jump_list=[45,50,30,50,40,40,40,50,35,40,45,45,40,30]
224 #labels to assign to each action
225 labels_list=[0,1,2,3,4,5,6,7,8,9,10,11,12,13]
226 #Get current directory
227 current_directory=os.getcwd()
228 #retrieve_desired_actions(current_directory,folders_to_explore,jump_list,
229 #                            jump=3)
230
231 transform_cloud_points_into_single_file_voxels(current_directory,\n
232 folders_to_explore,labels_list,voxgrid_size=[32,32,32])

```

Listing 6: Preprocessing: transform point clouds to voxels for multiple folders and delete the initial frames and smoothing of the frames

```

1 #Import standard libraries
2 import os
3 import numpy as np
4 import pandas as pd
5 import sys
6 import csv
7 import requests
8 import xml.etree.ElementTree as ET
9 import xmltodict
10 import shutil
11 import h5py
12
13 #glob lists the elements in the current directory with a specific pattern
14 import glob
15 from matplotlib import pyplot as plt
16 import matplotlib.patches as patches
17 from voxelgrid import VoxelGrid
18 from mpl_toolkits.mplot3d import Axes3D
19 from sklearn import preprocessing
20 from sklearn.model_selection import train_test_split
21
22 def get_list_elements_without_pattern_not_current_directory(
23     directory_to_search):
24     #Comprehension list that by given a directory, explores
25     files= [element for element in os.listdir(directory_to_search)]
26     #
27     return files

```

```
27
28 def create_folder_in_path_check_folder_created(path_creation,
29     path_to_create):
30     #
31     directories_in_directory_where_eant_create=\
32         get_list_elements_without_pattern_not_current_directory(path_creation)
33     #
34     directories_path_in_directory_where_eant_create=\
35         [path_creation+ '/' +path for path in \
36             directories_in_directory_where_eant_create]
37     #
38     if path_to_create not in
39         directories_path_in_directory_where_eant_create:
40             #
41                 os.mkdir(path_to_create)
42
43 #fucntion to create folders in a given path
44 def create_folders_in_path(path, folder_names_list):
45     #create folders in a given path. The folders names are given by a
46     #list
47     for folder_name in folder_names_list:
48         path_new_directory= path+ '/' + folder_name
49         create_folder_in_path_check_folder_created(path,
50             path_new_directory)
51
52
53 def merge_hpy_file(results_root_directory ,path_folders_with_files ,
54     list_folders_information_merge ,\
55     reduce_set=False , percentage_to_reduce_dataset=0.80):
56     #
57     paths_to_explore=[path_folders_with_files+ '/' +folder_analysis+ \
58         '_point_cloud' for \
59             folder_analysis in list_folders_information_merge]
60     #
61     directory_to_create_results=results_root_directory+ '/' +'merged_data'
62     #
63     create_folder_in_path_check_folder_created(results_root_directory ,\
64         directory_to_create_results)
65     #
66     number_paths_to_explore=len(paths_to_explore)
67     #
68     path_to_explore=paths_to_explore[0]
69     #
70     file_name=list_folders_information_merge[0]
71     #
72     file_name_path=path_to_explore+ '/' +file_name+ '.h5'
73     #
74     hf = h5py.File(file_name_path , 'r')
75     #
76     attributes= np.array(hf.get('attributes'))
77     #
78     labels=np.array(hf.get('labels'))
```

```
73     #
74     hf.close()
75     #
76     for path_to_explore_index in range(1,number_paths_to_explore):
77         #
78         path_to_explore=paths_to_explore[path_to_explore_index]
79         #
80         file_name=list_folders_information_merge[path_to_explore_index]
81         #
82         file_name_path=path_to_explore+ '/' +file_name + '.h5'
83         #
84         hf = h5py.File(file_name_path , 'r')
85         #
86         attributes_to_concatenate=np.array(hf.get('attributes'))
87         #
88         attributes=np.concatenate((attributes,attributes_to_concatenate),
axis=0)
89         #
90         labels_to_add=np.array(hf.get('labels'))
91         #
92         labels=np.concatenate((labels,labels_to_add), axis=0)
93         #
94         hf.close()
95
96     if reduce_set == True:
97         #
98         attributes_to_maintain,attributes_to_delete,\n99         labels_to_maintain,labels_to_delete=\n100        train_test_split(attributes,labels,\n101        test_size=percentage_to_reduce_dataset,stratify=labels,\nrandom_state=42)
102         #
103         percentage_data_kept=1-percentage_to_reduce_dataset
104         print (attributes_to_maintain.shape)
105         print (labels_to_maintain.shape)
106         print (np.unique(labels_to_maintain))
107         print (np.unique(labels_to_maintain, return_counts=True)[1])
108         #
109         attributes_training,attributes_testing,\n110         labels_training,labels_testing=\n111         train_test_split(attributes_to_maintain,labels_to_maintain,\n112         test_size=0.20,stratify=labels_to_maintain,\nrandom_state=42)
113         #
114         with h5py.File(directory_to_create_results+ '/' +'merged_dataset_',
+\n115             str(round(percentage_data_kept ,3))+ 'labelled_instances' + '.h5' , 'w'
') as hf:
116             hf.create_dataset('attributes_training' , data=
attributes_training)
117             hf.create_dataset('labels_training' , data=labels_training)
118             hf.create_dataset('attributes_testing' , data=
attributes_testing)
```

```
120         hf.create_dataset('labels_testing', data=labels_testing)
121     hf.close()
122     print ('training')
123     print (attributes_training.shape)
124     print ('testing')
125     print (labels_training.shape)
126     print (np.unique(labels_training))
127     print (np.unique(labels_training, return_counts=True)[1])
128     #
129     return 'done'
130
131 else:
132     #
133     attributes_training, attributes_testing, labels_training,
134     labels_testing=\
135     train_test_split(attributes, labels, test_size=0.20, stratify=
136     labels, \
137     random_state=42)
138     #
139     print (attributes_training.shape)
140     print (labels_training.shape)
141     print (np.unique(labels_training))
142     print (np.unique(labels_training, return_counts=True)[1])
143     #
144     with h5py.File(directory_to_create_results+'/'+'merged_dataset'+'.
145 .h5', \
146     'w') as hf:
147         hf.create_dataset('attributes_training', data=
148         attributes_training)
149         hf.create_dataset('labels_training', data=labels_training)
150         hf.create_dataset('attributes_testing', data=
151         attributes_testing)
152         hf.create_dataset('labels_testing', data=labels_testing)
153     hf.close()
154
155     return 'done'
156
157 """
158 def create_training_testing_sets(hpy_file_path, path_store_splited_file):
159     #
160     hf = h5py.File(hpy_file_path, 'r')
161     #
162     attributes= np.array(hf.get('attributes'))
163     print (attributes.shape)
164     #
165     labels=np.array(hf.get('labels'))
166     print(labels.shape)
167     #
168     attributes_training, attributes_testing, labels_training, labels_testing
169     =\
170     train_test_split(attributes, labels, test_size=0.15, stratify=labels, \
171     random_state=42)
172     #
```

```

167     hf.close()
168     #
169     with h5py.File(path_store_split_file+'+'trainig_testing_dataset
170     +''.h5', \
171     , 'w') as hf:
172         return print('done')
173     ''
174
175
176 ##### IMPLEMENTATION
177 #####
178 #Folder to explore
179 folders_to_explore=['punching','running_on_spot','chicken_wings','hips',\
180 'knees','jumping_jacks','shake_arms','shake_shoulders','shake_hips',\
181 'one_leg_loose','one_leg_jump','light_hopping_loose','light_hopping_stiff
182 ,\
183 'jiggle_on_toes']
184
185 #Jump list
186 jump_list=[45,50,30,50,40,40,40,50,35,40,45,45,40,30]
187 #labels to assign to each action
188 labels_list=[0,1,2,3,4,5,6,7,8,9,10,11,12,13]
189 #Get current directory
190 current_directory=os.getcwd()
191 #Create path to explore
192 path_explore=current_directory+'+'results_pointclouds'
193 #
194
195 merge_hpy_file(current_directory,path_explore,folders_to_explore)
196 #
197 splits=[0.8, 0.6, 0.4, 0.2]
198 for split in splits:
199     print (split)
200     merge_hpy_file(current_directory,path_explore,folders_to_explore, \
201     reduce_set=True, percentage_to_reduce_dataset=split)

```

Listing 7: Preprocessing: create training and testing sets

```

1 ## Code retrieved from https://www.kaggle.com/roestigraben/grid-of-voxels
2   -to-train-linear-model
3
4 import numpy as np
5 import open3d as o3d
6 import h5py
7 from mpl_toolkits.mplot3d import Axes3D
8 import matplotlib.pyplot as plt
9 from voxelgrid import VoxelGrid
10 import random
11 import os
12
13 def get_list_elements_without_pattern_not_current_directory(
14     directory_to_search):
15     #Comprehension list that by given a directory, explores

```

```
14     files= [element for element in os.listdir(directory_to_search)]
15     #
16     return files
17
18 def create_folder_in_path_check_folder_created(path_creation,
19     path_to_create):
20     #
21     directories_in_directory_where_eant_create=\
22     get_list_elements_without_pattern_not_current_directory(path_creation)
23     #
24     directories_path_in_directory_where_eant_create=\
25     [path_creation+'/'+path for path in \
26     directories_in_directory_where_eant_create]
27     #
28     if path_to_create not in
29     directories_path_in_directory_where_eant_create:
30         #
31         os.mkdir(path_to_create)
32
33 #fucntion to create folders in a given path
34 def create_folders_in_path(path, folder_names_list):
35     #create folders in a given path. The folders names are given by a
36     #list
37     for folder_name in folder_names_list:
38         path_new_directory= path+'/' + folder_name
39         create_folder_in_path_check_folder_created(path,
40         path_new_directory)
41
42 def reduce_dimension_point_cloud(point_cloud_to_reduce):
43     #create a od3 object
44     point_cloud = o3d.PointCloud()
45     #with the od3 trasnform the point cloud array into a od3 numpy array
46     point_cloud.points = o3d.Vector3dVector(point_cloud_to_reduce)
47     #o3d.draw_geometries([point_cloud])
48     #reduce the dimension of the point cloud
49     reduced_point_cloud= o3d.geometry.voxel_down_sample(point_cloud,
50     voxel_size=0.035)
51     #o3d.visualization.draw_geometries([reduced_point_cloud])
52     #tranform the 3od object back into a numpy array
53     reduced_point_cloud= np.asarray(reduced_point_cloud.points)
54     #return the reduced point cloud
55     return np.array(reduced_point_cloud)
56
57
58 #function to polish the shape of the point clouds
59 def modify_randomly_point_dimensions(point_cloud,dimension):
60     difference_point_cloud_dimensions=dimension-point_cloud.shape[0]
61     if difference_point_cloud_dimensions < 0:
62         difference_point_cloud_dimensions=-
63         difference_point_cloud_dimensions
64         #generate random numbers between 0 and the dimension of the point
65         cloud to
```

```
59     #modify
60     random_instances=random.sample(range(0,point_cloud.shape[0]),\
61         int(difference_point_cloud_dimensions))
62     #delete the instances randomly selected
63     normalised_point_cloud=np.delete(point_cloud,random_instances,
axis=0)
64
65 else:
66     #generate random numbers between 0 and the dimension of the point
cloud to
67     #modify
68     random_instances=random.sample(range(0,point_cloud.shape[0]),\
69         int(difference_point_cloud_dimensions))
70     #retrieve the random instances from the point cloud
71     retrieved_instances=point_cloud[random_instances]
72     #concatenate the retrieved instances to the the point cloud
73     normalised_point_cloud=\
74         np.concatenate((point_cloud,retrieved_instances),axis=0)
75     #return the normalised point cloud
76     return normalised_point_cloud
77
78 #function to transform an entire array/dataframe
79 def reduce_point_cloud_dataset(dataset):
80     #get the number of instances in the dataset
81     number_instances_dataset=dataset.shape[0]
82
83     #loop through all the instance in the dataset
84     for element_to_process_index in range(number_instances_dataset):
85         #First we retrieve the first element in the dataset and the we
86         #concatenate more elements to it
87         if element_to_process_index ==0:
88             #get the point cloud to reduce dimensionallity
89             element_to_process=dataset[element_to_process_index]
90             #reduce the dimension of the point cloud
91             reduced_point_cloud=reduce_dimension_point_cloud(
element_to_process)
92             #polish the shape of the point cloud
93             reduced_point_cloud=\
94                 modify_randomly_point_dimensions(reduced_point_cloud,1800)
95             # get the dimensions of the reduced point cloud
96             rows_point_reduced_cloud=reduced_point_cloud.shape[0]
97             columns_point_reduced_cloud=reduced_point_cloud.shape[1]
98             #reshape the point cloud in a way that we can concatenate
more
99             #point clouds to it
100            final_point_cloud_array=\
101                reduced_point_cloud.reshape\
102                    ((1,rows_point_reduced_cloud,columns_point_reduced_cloud))
103            #if is not the first element just append instances to the
original
104            else:
105                element_to_process=dataset[element_to_process_index]
106                #get the point cloud into a dataframe
```

```
107     reduced_point_cloud=reduce_dimension_point_cloud(
108         element_to_process)
109         #polish the shape of the point cloud
110         reduced_point_cloud=\
111             modify_randomly_point_dimensions(reduced_point_cloud,1800)
112             # get the dimensions of the reduced point cloud
113             rows_point_reduced_cloud=reduced_point_cloud.shape[0]
114             columns_point_reduced_cloud=reduced_point_cloud.shape[1]
115             #reshape the point cloud and concatenate it to the main
116             results
117             #structure
118             reduced_point_cloud=\
119                 reduced_point_cloud.reshape((1,\\
120                     rows_point_reduced_cloud,columns_point_reduced_cloud))
121                     #concatenation
122                     final_point_cloud_array=\
123                         np.concatenate((final_point_cloud_array,reduced_point_cloud),
124                         axis=0)
125
126     #return the entire modified set
127     return final_point_cloud_array
128
129 ##### IMPLEMENTATION
130 #####
131
132 #get current directory
133 current_directory=os.getcwd()
134 #directory to store results
135 results_directory=current_directory+ '/' + 'processed_point_clouds'
136
137 #create a directory to store the processed point clouds
138 create_folder_in_path_check_folder_created(current_directory,
    results_directory)
139
140 #directory with the data
141 data_directory=current_directory+ '/' + 'merged_data'
142
143 datafiles_to_process=['merged_dataset.h5', \
144 'merged_dataset_0.8labelled_instances.h5', \
145 'merged_dataset_0.6labelled_instances.h5', \
146 'merged_dataset_0.4labelled_instances.h5', \
147 'merged_dataset_0.2labelled_instances.h5']
148
149 datafiles_name=['full_pt','0.8 dataset_pt','0.6 dataset_pt','0.4
    dataset_pt', \
150 '0.2 dataset_pt']
151
152 #get the number of databases
153 number_datasets=len(datafiles_to_process)
154
155 #loop through all the data
156 for dataset_index in range(number_datasets):
    #get the dataset name
```

```
154     dataset_name= datafiles_to_process[dataset_index]
155
156     #get the data
157     dataset= h5py.File(data_directory+ '/' +dataset_name , 'r')
158     attributes_training=np.array(dataset.get('attributes_training'))
159     attributes_testing=np.array(dataset.get('attributes_testing'))
160     labels_training=np.array(dataset.get('labels_training'))
161     labels_testing=np.array(dataset.get('labels_testing'))
162     dataset.close()
163
164     #sanity check with print statements
165     print (attributes_training.shape)
166
167     #process the datasets
168     attributes_training= reduce_point_cloud_dataset(attributes_training)
169     attributes_testing= reduce_point_cloud_dataset(attributes_testing)
170
171     #sanity check with print statements
172     print (attributes_training.shape)
173
174     #save the dataset
175     with h5py.File(results_directory+ '/' +datafiles_name[dataset_index]+ '.h5' \
176     , 'w') as hf:
177         hf.create_dataset('attributes_training' , data=attributes_training)
178         hf.create_dataset('labels_training' , data=labels_training)
179         hf.create_dataset('attributes_testing' , data=attributes_testing)
180         hf.create_dataset('labels_testing' , data=labels_testing)
181         hf.close()
```

Listing 8: Algorithm point clouds to voxels