People's Democratic Republic of Algeria Ministry of Higher Education and Scientific Research
Saad Dahleb Blida University – Blida 1
Computer Science department

3D Shape generation from a short text description

In order to obtain the Master's degree
Domain: Mathematics and computer science
Branch: Natural Language Processing

Realized by:
Selmane Mohammed Ayyoub
Miloudi Merouane

Supervised by:
Mr. Kameche Abdallah Hicham

In front of the jury:
Mr. Cherif Zahar Sid Ahmed Amine
Mrs. Berramdane Djamila

Date: _ / _ / _

Year: 2020 - 2021

Abstract:

Today, the world witnesses a huge advancement in technology, specially in the domain of virtual (VR) and augmented reality (AR). A lot of the largest companies are interested in AR and VR, an imaginary world that you feel inside. That kind of project are based on visualization of 3d objects. It was always considered a difficult task for designers to build a full 3D environment. Even if it is possible to do it, it could not be achieved neither in a short time nor with less expensive software

The objective of this work is to generate a 3d shape from a short text description with the help of the most interesting topic in IA neural, networks and specifically, the generative adversarial networks (GAN).

We built and trained a conditional GAN (CGAN) having as input the Bert embeddings of text descriptions and as output their corresponding 3D shape embeddings. We trained an Autoencoder to learn the 3D shapes representations. To validate our model, we trained another CGAN having as output the 3D Shapes. We noticed that there isn't a big loss in the obtained 3D forms.

Keywords: Text to 3D shape, Text embeddings, Shape embeddings, CGAN, Bert, Autoencoder.

Résumé :

Aujourd'hui, le monde est témoin d'un énorme progrès technologique, en particulier dans le domaine de la réalité virtuelle et augmentée. De nombreuses grandes entreprises s'intéressent à la réalité augmentée et à la réalité virtuelle. Ce genre de projet est basé sur la visualisation d'objets 3D. Concevoir un environnement 3D a toujours été considéré comme une tâche difficile pour les concepteurs. Et même si c'est possible de le faire, cela ne peut être réalisé ni en peu de temps ni avec des logiciels moins coûteux.

L'objectif de ce travail est de générer une forme 3D à partir d'une description textuelle à l'aide du sujet le plus intéressant en IA les réseaux de neurones et plus particulièrement les réseaux antagonistes génératifs (GAN).

Nous avons construit et entraîné un GAN conditionnel (CGAN) ayant en entrée les représentations Bert des descriptions textuelles et en sortie leurs représentations de forme 3D correspondantes. Pour apprendre les représentations des formes 3D, nous avons entrainé un auto encodeur. Pour valider notre modèle, nous avons entrainé un autre CGAN ayant en sortie les Formes 3D. Nous avons remarqué qu'il n'y a pas de perte significative dans les formes 3D obtenues.

Mots-clés : : Text to 3D shape, Représentation Textuelle, représentation des formes 3D, CGAN, Bert, Autoencoder.

ملخص:

يشهد العالم اليوم تقدمًا هائلاً في التكنولوجيا خاصة في مجال الواقع الافتراضي والواقع المعزز، الكثير من الشركات مهتمة بالواقع المعزز والواقع الافتراضي، عالم خيالي تشعر وكأنك بداخله. هذا النوع من المشروع الذي يعتمد على تصور اشياء ثلاثية الأبعاد، كان يُعتبر دائمًا مهمة صعبة للمصممين، حتى لو كان من الممكن القيام بذلك، فلا يمكن تحقيقه لا في وقت قصير ولا باستخدام برامج أقل تكلفة. وبما أننا طلاب في تخصص الذكاء الاصطناعي، فقد فكرنا في كيف يمكننا المساعدة في مثل هذه المشكلة.

الهدف من هذا العمل هو إنشاء شكل ثلاثي الأبعاد من وصف النص بمساعدة الموضوع الأكثر إثارة للاهتمام في الذكاء الصناعي الشبكات العصبية وعلى وجه التحديد شبكات الخصومة التوليدية.

الكلمات الرئيسية: التعلم الآلي، الشبكات العصبية، أوصاف النص، الأشكال ثلاثية الأبعاد، تحويل النص إلى شكل ثلاثي الابعاد، GAN، بيرت.

Acknowledgements

Above all, it is thanks to Allah that we were able to get to where we are. It is above all thanks to Him that our path has been lit to finish our modest work.

And then to those who helped us from the verry beginning till the end of this work. On top of that of course it is our instructor Mr. Kameche, who gave us this huge opportunity to get some experience handling real life problems, and helped us through it all while teaching us some valuable that we wouldn't have had the slightest of chance to come across anywhere else genuinely thank you monsieur. Our parents who have been there for us all along, you have sacrificed everything for your children sparing neither health nor efforts. You have given us a wonderful model of perseverance. We are forever in your debt.

We would like to express our gratitude to our brothers and sisters, family, and friends for their encouragement.

We couldn't have finished the acknowledgement part without mentioning Mme. Mezzi, we have only good memories by your side. Thank you Madame for being who you are, passionate about your work, thank you for offering your time to help us even outside of your working hours. We will never forget that your sessions were both productive and fun. You are an example to us.

To all of them, we extend our thanks, respect, and gratitude.

General introduction:

Context and problematic

Artificial intelligence and machine learning have taken the world by storm in the last couple of years and the best seems is yet to come, and its bread and butter is the data, a need for a smooth transition between different types of data haven't been more necessary. Although there have been many works on that but the 3d object type of data didn't seem to be getting its fair share of interest? In all fairness, it makes all the sense considering how complex time and hardware (money) consuming they are. Companies as big as facebook and amazon started to invest in advanced reality(AR) related fields which we think is due to the advance in computation algorithms, which led into more people believing it is becoming a good future investment.

Along the way, some issues were raised. Is it possible to learn a 3D shape representation? Is there a way to automatically generate 3D AR environment from text descriptions?

Work objectives

Our work is nothing short of those algorithms where we take on the problem of 3D objects and try to find the most cost-efficient way of using them but on the other hand also doesn't affect the results (or affect them in an acceptable way not that big to feel a difference).

Our mains tasks are:

- Build a neural model to learn 3D shapes representations (embeddings)
- Build a model that associate 3D shapes and text descriptions using only their representations
- Validate the final model using a real dataset

We will use a labeled dataset of tables and chairs with textual descriptions and try to develop some deep learning algorithms to train a model on them that will, in the end, understand and be able to perform cross modal (3D objects or text) tasks like generation or classification.

We divided our work into a theoretical and a practical part

Chapter 1: where we will be explaining machine learning and showing how magical their algorithms can be which shows why we chose them to handle our problem.

Chapter 2: in this chapter we will discuss the similar works that are connected in a way or another to our work

That concludes the theoretical part, and in the practical one we have.

Chapter 3: here we will detail the conceptual part of our work, where we explain our deep learning models and how they work as well as their architectures and why we chose every single one of them.

Chapter 4: lastly we discuss the implementation of those models and what parameters did we use to get the best out of them, then exploring the non-technical stuff that is just as important, to finish off with an evaluation to all the deep learning model

Contents

List of Figures

CHAPTER 1

CHAPTER 2

CHAPTER 1

A GLOBAL Overview On MACHINE LEARNING

1.1 INTRODUCTION

Machine learning is an evolving branch of computational algorithms that are designed to emulate human intelligence by learning from the surrounding environment, it helps computers understand brute data and make decisions or even predictions (which can be even better than humans) based on what they have learned at first. It is,in other words, the field of study that gives computers the ability to learn without being explicitly programmed. These days, techniques based on machine learning have been applied successfully in diverse fields ranging from pattern recognition, computer vision, spacecraft engineering, finance, entertainment, and computational biology to biomedical and medical applications.

1.2 MACHINE LEARNING

At its most basic, machine learning uses programmed algorithms that receive and analyse input data to predict output values within an acceptable range. As new data is fed to these algorithms, they learn and optimise their operations to improve performance, developing 'intelligence' over time.

There are four types of machine learning algorithms: supervised, semi-supervised, unsupervised and reinforcement.

1.2.1 Supervised LEARNING

Supervised learning is the type of machine learning in which machines are trained using well "labelled" training data, and on basis of that data, machines predict [1]. The labelled data means some input data is already tagged with the correct output. Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function between the input and the output. Y=F(X). The goal is to approximate the

mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data In supervised learning, the training data provided to the machines work as the supervisor that teaches the machines to predict the output correctly. It applies the same concept as a student learns in the supervision of the teacher. In the real-world, supervised learning can be used for Risk Assessment, Image classification, Fraud Detection, spam filtering, etc. The working of Supervised learning can be easily understood by the below example and diagram:

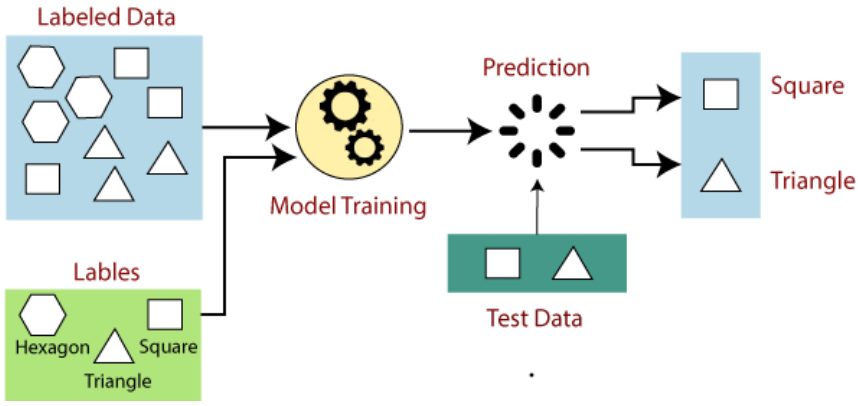

FIGURE 1.1. supervised learning [54]

Supervised learning can be further divided into two types of problems [2]:

- Regression, used for the prediction of continuous variables, such as Weather forecasting, Market Trends.
- Classification, used when the output variable is categorical, which means there are two classes such as Yes-No, Male-Female, True-false.

1.2.2 *Unsupervised LEARNING*

Unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and insights from the given data. It can be compared to learning which takes place in the human brain while learning new things. It can be defined as machine learning in which models are trained using unlabeled dataset and are allowed to act on that data without any supervision. Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output

data. The goal of unsupervised learning is to find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format. In real-world, we do not always have input data with the corresponding output so to solve such cases, we need unsupervised learning. Rewards of unsupervised learning can be understood by the below diagram:

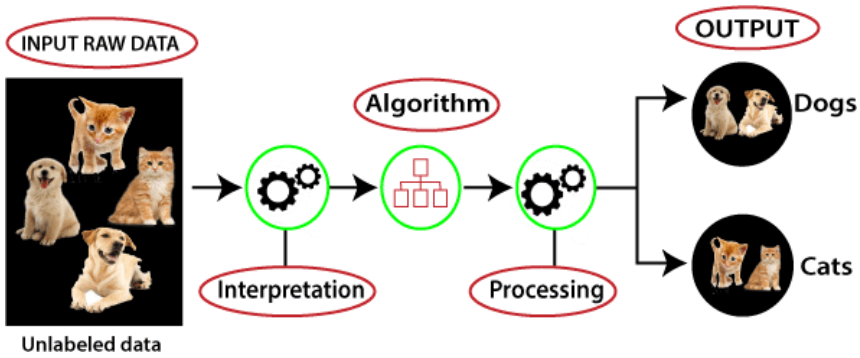

FIGURE 1.2. unsupervised learning [55]

The unsupervised learning algorithm can be further categorized into two types of problems:

- Clustering, it is a method of grouping the objects into clusters such that objects with most similarities remains into a group and has less or no similarities with the objects of another group.
- Association, it used for finding the relationships between variables in the large database. It determines the set of items that occurs together in the dataset. Association rule makes marketing strategy more effective. Such as people who buy X item (suppose a bread) are also tend to purchase Y (Butter/Jam) item.

1.2.3 *semi-supervised LEARNING*

Semi-Supervised learning is a type of Machine Learning algorithm that represents the intermediate ground between Supervised and Unsupervised learning algorithms. It uses the combination of labeled and unlabeled datasets during the training period. The basic disadvantage of supervised learning is that it requires hand-labeling by ML specialists or data scientists, and it also requires a high cost to process. Further unsupervised learning also has a limited spectrum

for its applications. To overcome these drawbacks of supervised learning and unsupervised learning algorithms, the concept of Semi-supervised learning is introduced. In this algorithm, training data is a combination of both labeled and unlabeled data. However, labeled data exists with a very small amount while it consists of a huge amount of unlabeled data. Initially, similar data is clustered along with an unsupervised learning algorithm, and further, it helps to label the unlabeled data into labeled data uses pseudo labeling to train the model with less labeled training data than supervised learning Semi-supervised learning models are becoming more popular in the industries, some of the main applications are: Speech Analysis, Web content classification, Protein sequence classification.[2]

1.2.4 Reinforcement LEARNING

Reinforcement Learning is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions. For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty. The agent learns automatically using feedbacks without any labeled data, and since there is no labeled data, so the agent is bound to learn by its experience only and interacts with the environment and explores it by itself. The primary goal of an agent in reinforcement learning is to improve the performance by getting the maximum positive rewards Reinforcement Learning solves a specific type of problem where decision making is sequential, and the goal is long-term, such as game-playing, robotics, etc. Rewards and reinforcement learning can be understood by the below diagram:

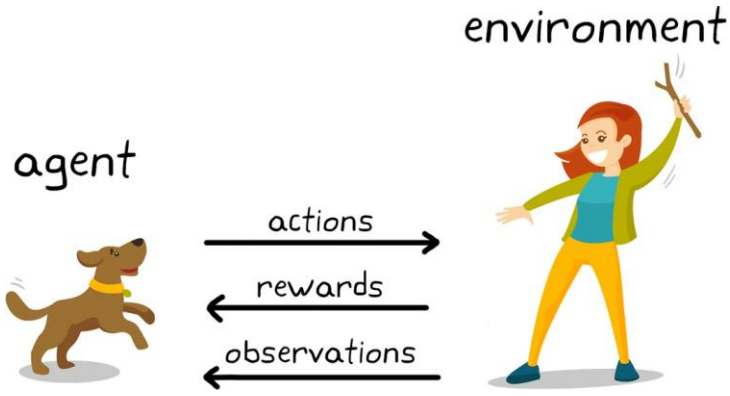

FIGURE 1.3. reinforcement learning [56]

1.3 DEEP LEARNING

Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers. Neural networks are just one of many tools and approaches used in machine learning algorithms. An artificial neural network is usually a computational network based on biological neural networks that construct the structure of the human brain. Artificial neural networks have neurons that are linked to each other in various layers of the networks. These neurons are known as nodes. Every neural network must have at least three layers of neurons which are the input, the output and the hidden layer where every neuron in a given layer is fully connected with the layer that precedes (to pass from a neuron of the input layer to a neuron from the hidden layer you must pass by the arc that is associated with a weight and every layer has its own activation function Y = F(x*w) where Y is the output X is the input, w is the weight of the arc and F is the activation function) this step is called the forward propagation.[3]

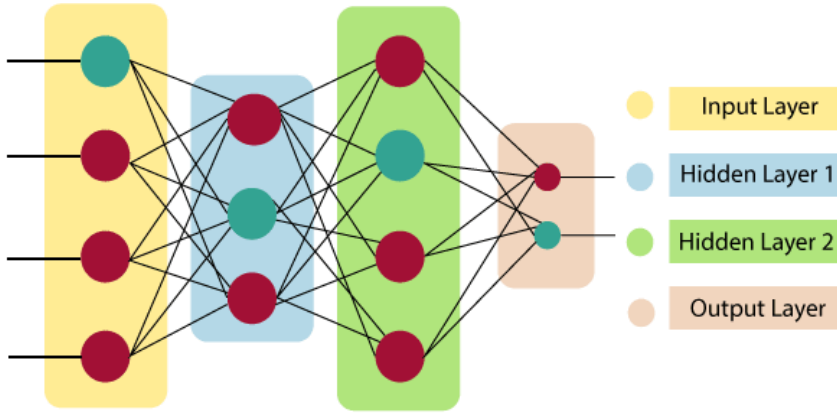

FIGURE 1.4. deep learning [57]

1.3.1 *ACTIVATION functions*

Neural networks are specifically designed based on the inner workings of biological brains. These models imitate the functions of interconnected neurons by passing input features through several layers of what are referred to as perceptrons (neurons), each transforming the input using a set of functions. This section will tackles the different modules that make this possible. The activation function refers to the set of transfer functions used to achieve the

desired output. We can classify activation functions in two categories :

(a) linear function

In the linear activation function, the output of functions is not restricted in between any range. Its range is specified from -infinity to infinity. For each individual neuron, the inputs get multiplied with the weight of each respective neuron, which in turn leads to the creation of output signal proportional to the input. If all the input layers are linear in nature, then the final activation of the last layer will actually be the linear function of the initial layer's input.[4]

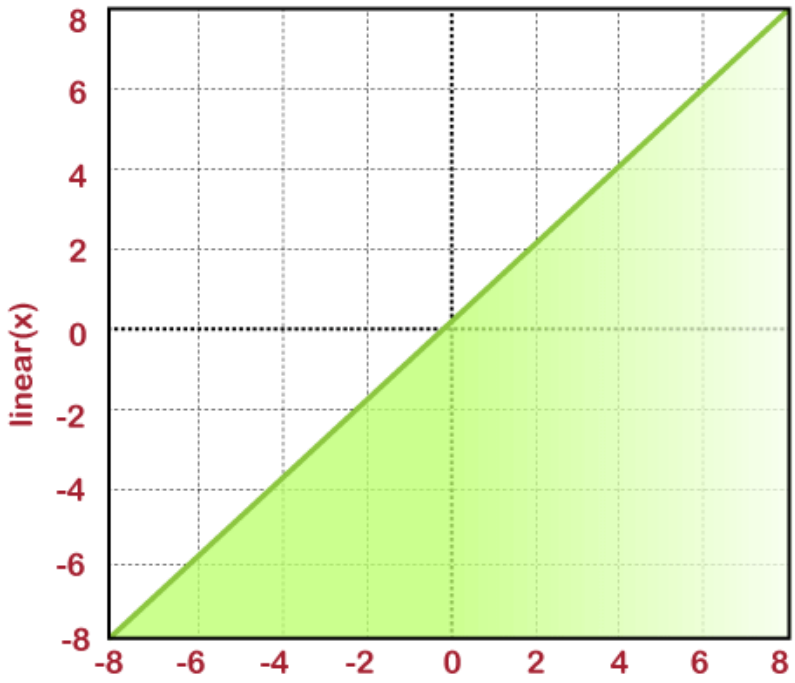

FIGURE 1.5. linear function [58]

(b) Non-Linear Function

These are one of the most widely used activation function. It helps the model in generalizing and adapting any sort of data in order to perform correct differentiation among the output. It solves the following problems faced by linear activation functions which is problems related to backpropagation and stacking up of several layers of the neurons. The non-linear activation function is further

divided into: Sigmoid or Logistic Activation Function, Tanhor Hyperbolic Tangent Activation Function, ReLU (Rectified Linear Unit) Activation Function and Softmax Function.[4]

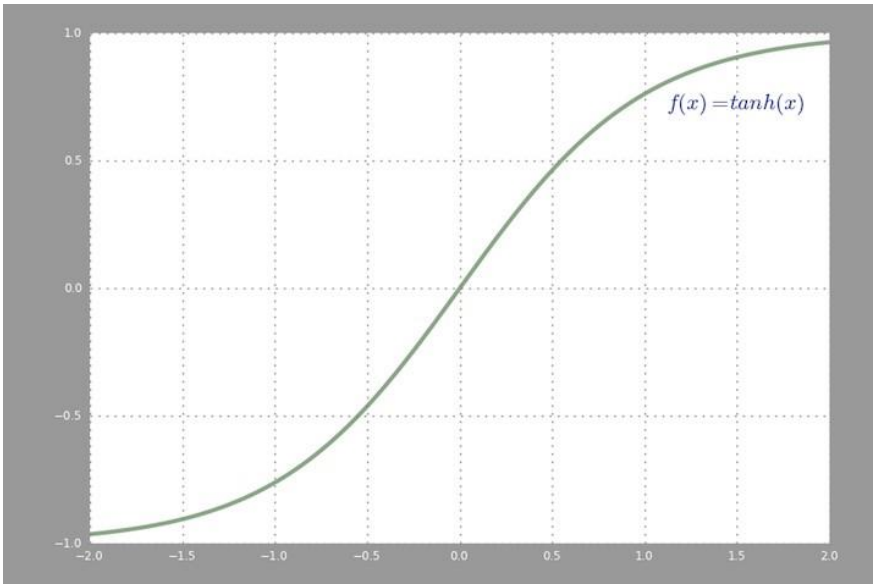

FIGURE 1.6. nonlinear function [59]

1.3.2 Loss Function

The loss function is attempting to minimize by continuously updating the weights in the model during training. During the training process, at the end of each epoch, the loss will be calculated on the models predictions. So basically what is happening is that the model calculates the error on each input by looking at what output it predicted for that input, and taking the difference of that output value and the correct label for that input. For example, if our model was classifying images of cats and dogs, then say the label for a cat is zero, and the label for a dog is one. If we pass an image of a cat to our model, and our model outputs 0.25 for this image, then the error between the models output versus the true label for the image would be 0.25 minus zero, the label for cat which is equal to 0.25. So it does this process for every input, then at the end of each epoch, it will accumulate all of the individual errors for each input, and then in some way, pass them through to a loss function.[4]

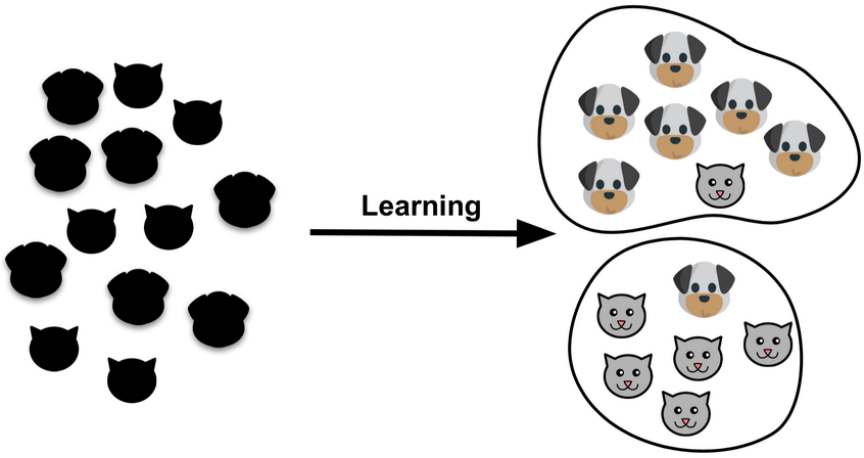

FIGURE 1.7. cats dogs classification example [60]

1.3.3 GRADIENT Decent

The Gradient Descent is an optimization algorithm which is used to minimize the cost function for many machine learning algorithms. Gradient Descent algorithm is used for updating the parameters of the learning models. There are different types of Gradient Descent.[4]

1. Batch Gradient Descent

   The Batch Gradient Descent is the type of Gradient Algorithm that is used for processing all the training datasets for each iteration of the gradient descent. Suppose the number of the training dataset is large, the batch gradient descent will be comparatively expensive. Hence, if the number of the training dataset is large, the users are not advised to use batch gradient descent. Instead, they can use mini-batch gradient descent for a large training dataset.[4]

2. Mini-Batch Gradient Descent

   The mini-batch gradient descent is the type of gradient descent that is used for working faster than the other two types of gradient descent. Suppose the user has 'p' (where 'p' is batch gradient descent) dataset where $p < m$ (where 'm' is mini-batch gradient descent) will be processed per iteration. So, even if the number of 'p' training dataset is large, the mini-batch gradient descent will process it in batches of 'p' training datasets in a single attempt. Therefore, it can work for large training datasets with fewer numbers of iterations.[5]

3. Stochastic Gradient Descent

tochastic gradient descent is the type of gradient descent which can process one training dataset per iteration. Therefore, the parameters will be updated after each iteration, in which only one dataset has been processed. This type of gradient descent is faster than the Batch Gradient Descent. But, if the number of training datasets is large then also, it will process only one dataset at a time. Therefore, the number of iterations will be large.[4]

1.4 DEEP LEARNING ARCHITECTURES

Deep learning is the most advanced feat scientists have reached to this day, and that's because it uses some specific architectures to understand different types of complex data and make decisions, we will now be looking at some of teh most important of these architectures.

1.4.1 CONVOLUTIONAL NEURAL network

CNN is one of the techniques to do image classification and image recognition in neural networks (it can treat other input data but it is not as efficient like when working with images). It is designed to process the data by multiple layers of arrays. This type of neural network is used in applications like image recognition or face recognition. The primary difference between CNN and other neural networks is that CNN takes input as a two-dimensional array. And it operates directly on the images rather than focusing on feature extraction which other neural networks do.Convolutional Neural Network (CNN or ConvNet) is a type of feed-forward artificial networks where the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex.CNN takes an image as input, which is classified and process under a certain category such as dog, cat, lion, tiger, etc. The computer sees an image as an array of pixels and depends on the resolution of the image. Convolutional Neural Networks have the following 4 layers:

1. the convolutional layer

it is the first layer to extract features from an input image. By learning image features using a small square of input data, the convolutional layer preserves the relationship between pixels. It is a mathematical operation which takes two inputs such as image matrix and a kernel or filter.

- The dimension of the image matrix is **h×w×d**.
- The dimension of the filter is $\mathbf{f_h \times f_w \times d}$.
- The dimension of the output is $\mathbf{(h-f_h+1)\times(w-f_w+1)\times 1}$.

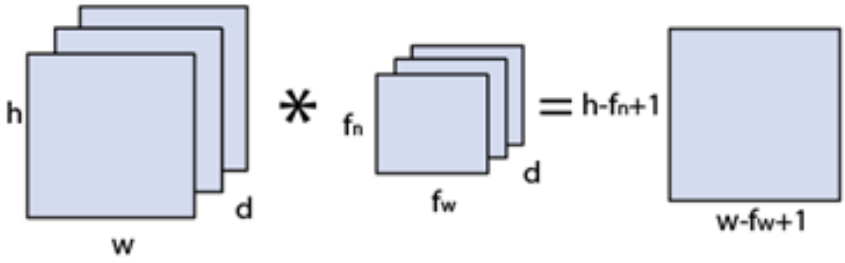

FIGURE 1.8. the convolutional layer [61]

2. Rectified Linear unit(ReLU)

it is a transform functions only activates a node if the input is above a certain quantity. While the data is below zero, the output is zero, but when the input rises above a certain threshold. It has a linear relationship with the dependent variable.In this layer, we remove every negative value from the filtered images and replaces them with zeros.It is happening to avoid the values from adding up to zero.

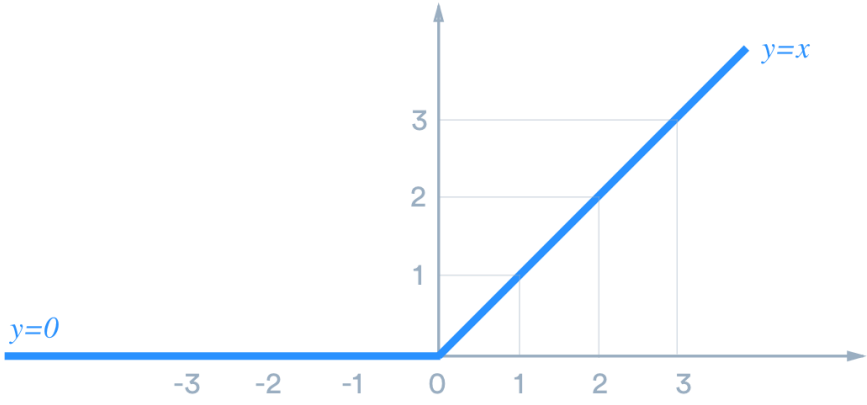

FIGURE 1.9. Rectified Linear unit [62]

3. the pooling layer

t plays an important role in pre-processing of an image. Pooling layer reduces the number of parameters when the images are too large. Pooling is "downscaling" of the image obtained from the previous layers. It can be compared to shrinking an image to reduce its pixel density. Spatial pooling is also called downsampling or subsampling, which reduces the dimensionality of each map but retains the important information.

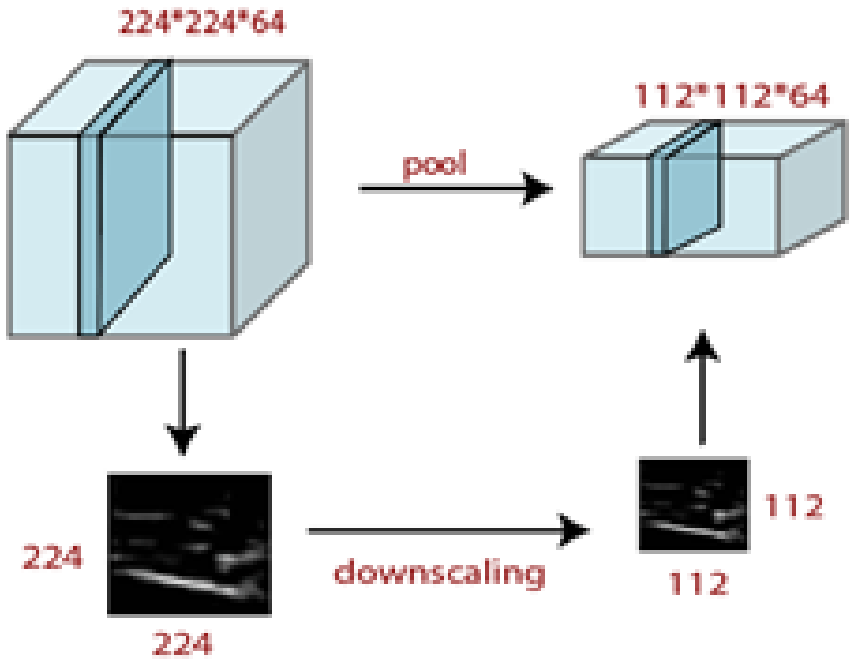

FIGURE 1.10. the pooling layer [63]

4. the fully connected layer

it is a layer in which the input from the other layers will be flattened into a vector and sent. It will transform the output into the desired number of classes by the network.In the above diagram, the feature map matrix will be converted into the vector such as X1, X2, X3... Xn with the help of fully connected layers. We will combine features to create a model and apply the activation function such as softmax or sigmoid to classify the outputs as a car, dog, truck, etc.

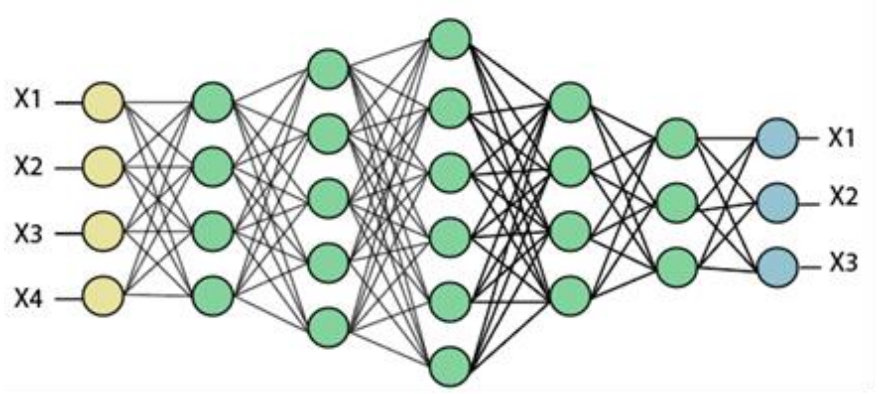

FIGURE 1.11. the fully connected layer [64]

1.4.2 Recurrent NEURAL networks

A recurrent neural network (RNN) is a kind of artificial neural network mainly used in speech recognition and natural language processing (NLP). RNN is designed to recognize patterns in sequences of data, such as text, genomes, handwriting, the spoken word, and numerical time series data emanating from sensors, stock markets, and government agencies. A recurrent neural network looks similar to a traditional neural network except that a memory-state is added to the neurons. The computation is to include a simple memory. The recurrent neural network is a type of deep learning-oriented algorithm, which follows a sequential approach. In neural networks, we always assume that each input and output is dependent on all other layers. These types of neural networks are called recurrent because they sequentially perform mathematical computations. Recurrent Neural Networks suffer from short-term memory. If a sequence is too long, they won't be able to carry all the important information from past steps. When processing a text to do predictions, RNNs may leave out important information from the beginning.

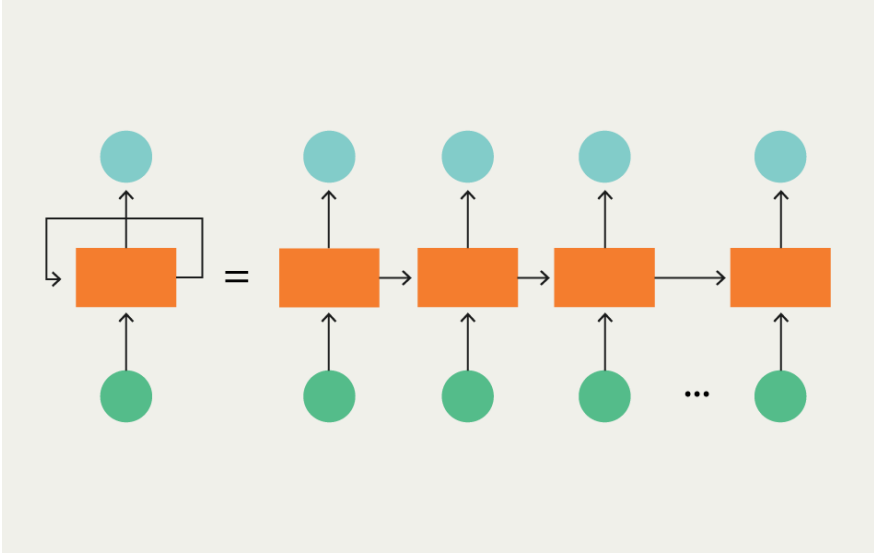

FIGURE 1.12. a recurrent neural network [65]

RNN is used in different types of models here are the most relevant ones:

1. Vector-Sequence Models

   They take fixed-sized vectors as inputs and output vectors of any length, for example, in image captioning, the image is given as an input and the output describes the image.

2. Sequence-Vector Model

   Take a vector of any size and output a vector of fixed size. Eg. Sentiment analysis of a movie rates the review of any movie as positive or negative as a fixed size vector.

3. Sequence-to-Sequence Model

   The most popular and most used variant, take input as a sequence and give output as another sequence with variant sizes. Example: Language translation, for time series data for stock market prediction.RNN disadvantages: slow to train and long sequence leads to vanishing gradient or, say, the problem of long term dependencies. In simple terms, its memory is not that strong when it comes to remembering old connections. Therefore they had to come with solutions, the best they made are :

   (a) LSTM:

   Long Short Term Memory- Special kind of RNN, specially made for solving vanishing gradient problems. They are capable of learning

Long-Term Dependencies. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn it.The LSTM Neurons have unlike normal neurons have a branch that allows to pass information and to skip the long processing of the current cell, this allows the memory to be retained for a longer period of time. It does improve the situation of the vanishing gradient problem but not that amazingly, like it will do good till 100 words, but for like 1,000 words, it starts to lose its grip.But like simple RNN it is also very slow to train, or even slower.LSTM take input sequentially one by one, which is not able to use up GPU's very well, which are designed for parallel computation.

1.4.3 TRANSFORMERS

Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution. Therefore we need to introduce attention first, we can explain attention in neural network by a simple example, supposing we have a book of machine learning and we need information about categorical cross-entropy. here are two ways of doing it, first, read the whole book and come back with the answer. Second, go to the index, find the 'losses' chapter, go to the cross-entropy part and read the part of Categorical Cross Entropy. In the former case we didn't focus on any part of the book specifically, whereas in the latter case, we focused our attention on the chapter of losses and then further focused our attention on the cross-entropy part where the concept of Categorical Cross Entropy is explained. Actually, this is the way most of us humans will do. Attention in neural networks is somewhat similar to what we find in humans. They focus on the high resolution in certain parts of the inputs while the rest of the input is in low resolution Remember hidden state from simple RNN now actually it is the context vector we pass along to the decoder. The context vector turned out to be problematic for these types of models. Models have a problem while dealing with long sentences. Or say they were facing the vanishing gradient problem in long sentences. So, a solution came along in a paper, Attention was introduced. It highly improved the quality of machine translation as it allows the model to focus on the relevant part of the input sequence as needed. Transformers are made to solve the problem of slow training by the input sequence can be passed parallelly so that GPU can be used effectively The Transformer starts by generating initial representations, or embeddings, for each word. These are represented by the

unfilled circles. Then, using self-attention, it aggregates information from all of the other words, generating a new representation per word informed by the entire context, represented by the filled balls. This step is then repeated multiple times in parallel for all words, successively generating new representations. The decoder operates similarly, but generates one word at a time, from left to right. It attends not only to the other previously generated words but also to the final representations generated by the encoder. Transformers gives wonderful results, using a self-attention mechanism and also solves the parallelization issue. Even Google uses BERT that uses a transformer to pre-train models for common NLP applications.

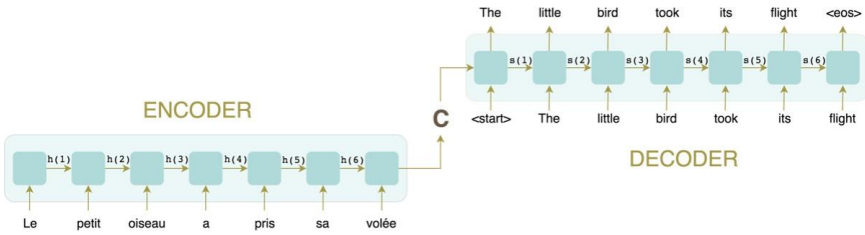

FIGURE 1.13. a transformer handling an NLP task [66]

1.4.4 Autoencodes

Auto-encoders are a special type of neural network where inputs are outputs are found usually identical. It was designed to primarily solve the problems related to unsupervised learning. Auto-encoders are highly trained neural networks that replicate the data. It is the reason why the input and output are generally the same. They are used to achieve tasks like pharma discovery, image processing, and population prediction. Auto-encoders constitute three components namely the encoder, the code, and the decoder. Auto-encoders are built in such a structure that they can receive inputs and transform them into various representations. The attempts to copy the original input by reconstructing them is more accurate. They do this by encoding the image or input, reduce the size. If the image is not visible properly they are passed to the neural network for clarification. Then, the clarified image is termed a reconstructed image and this resembles as accurate as of the previous image. To understand this complex process, see the below-provided image. They are mainly designed to encode the output into a compressed yet meaningful representation and then decode it back such that the reconstructed output is similar to the original one (the output). The problem is to learn the functions $A : R n \rightarrow R p$ (encoder) and

B : R p → R n (decoder) that satisfy

$$arg_{min}\ (a,\ b)E\ [(x,\ BA(x\ )]$$

where E is the expectation over the distribution of x, and is the reconstruction loss function, which measures the distance between the output of the decoder and the input. if A and B were linear operations we get a linear auto-encoder.

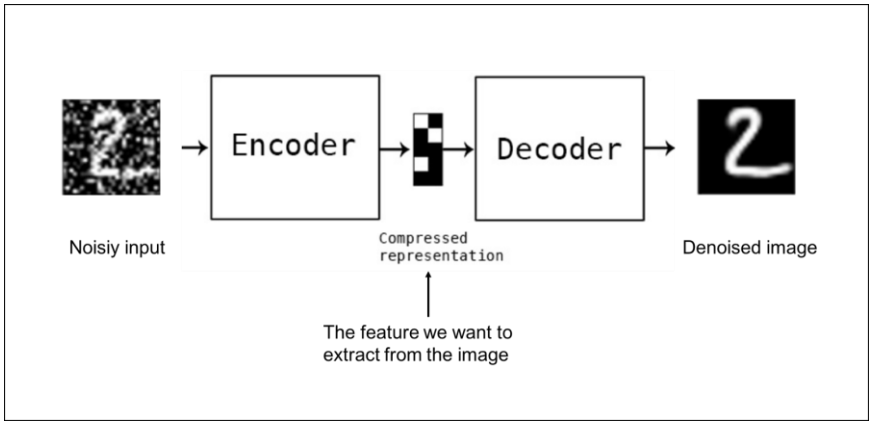

FIGURE 1.14. an example of an autoencoder [67]

1.4.5 GENERATIVE Architecture

1. Generative Adversarial Neural Networks

   Generative adversarial networks are a new but rapidly growing algorithmic architectures that uses two adversarial neural networks (they work one against the other), they are called the generator and the discriminator. The generator tries to fool the discriminator by generating data similar to those of the training set while the discriminator tries to identify fake from real data, working simultaneously they can learn and train complex data. The algorithm works as follows: the generator receives a random noise input and then transforms it into some meaningful output that is going to be fed to the discriminator alongside with the training set in order for it to try to identify the original training data from the generated ones, mathematically speaking, the discriminator and generator play a two-player minimax game with the value function V(G, D). So, Minimax Objective function is:

$$Y\ (D;G) = Ex \sim p\ (data)[logD\ (x\ )] + Es \sim p\ (s)[log(1\ -\ D\ (G\ (s)))]$$

where D(x) is the probability that data x is from the training set, and D(G(Z)) is the probability that data G(Z) ,which is fake data generated by the generator, are from the training set.This way the generator tries to minimize V (by minimizing D(x) and D(Z)), and the discriminator tries to maximize V(by maximizing D(x) and minimizing D(Z)).they both learn by alternative gradient descent (we apply GD on a neural net while fixing the other and vice versa).

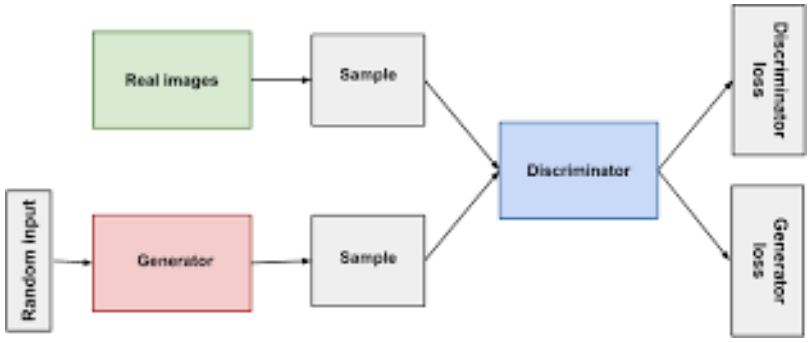

FIGURE 1.15. Generative Adversarial Network structure [68]

2. Conditional Generative Adversarial Networks

GANs can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information (say Y).Y could be any kind of auxiliary information,such as class labels or data from other modalities. We can perform the conditioning by feeding Y into the both the discriminator and generator as an additional input layer.The new objective function of the two-player minimax game would be:

$$Y\ (D;G) = Ex \sim p\ (data)[logD\ (x\ /y)]+Es \sim p\ (s)[log(1-D\ (G\ (s/y)))]$$

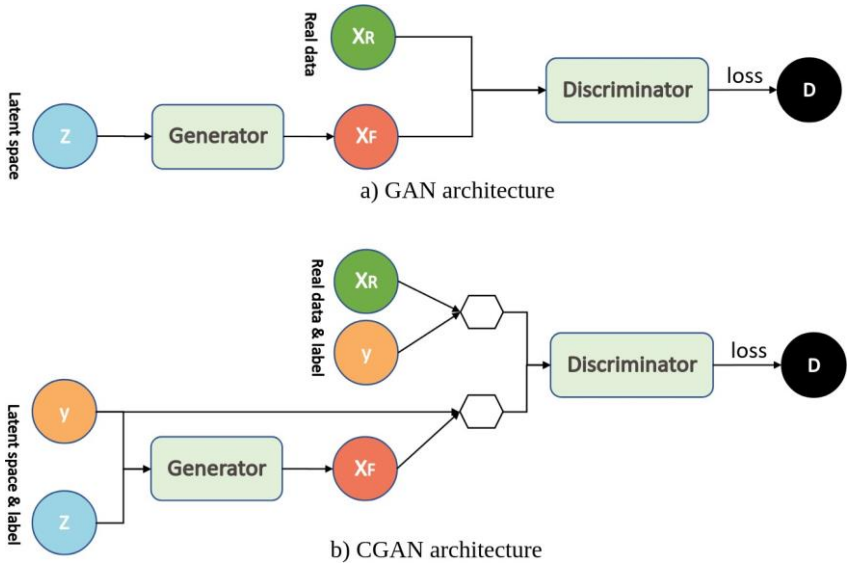

FIGURE 1.16. A comparison between GAN and CGAN [69]

3. Wasserstein Generative Adversarial Networks

The Wasserstein GAN or WGAN, was introduced by Martin Arjovsky, et al. in their 2017 paper [6]. It is an extension of the GAN that seeks an alternate way of training the generator model to better approximate the distribution of data observed in a given training dataset. Instead of using a discriminator to classify or predict the probability of generated images as being real or fake, the WGAN changes or replaces the discriminator model with a critic that scores the realness or fakeness of a given image. This change is motivated by a theoretical argument that training the generator should seek a minimization of the distance between the distribution of the data observed in the training dataset and the distribution observed in generated examples. The benefit of the WGAN is that the training process is more stable and less sensitive to model architecture and choice of hyperparameter configurations. Perhaps most importantly, the loss of the discriminator appears to relate to the quality of images created by the generator. The primary contribution of the WGAN model is the use of a new loss function that encourages the discriminator to predict a score of how real or fake a given input looks. This transforms the role of the discriminator from a classifier into a critic for scoring the realness or fakeness of images, where the difference between the scores is as large as possible.

1.5 MACHINE LEARNING MODELS PERFORMANCE TESTING

The model training is an important step, but after that step how the model generalizes on unseen data is an equally important aspect that should be considered in every machine learning pipeline. We need to know whether it actually works and, consequently, if we can trust its predictions in future. This issues can be handled by evaluating the performance of a machine learning model, Model evaluation aims to estimate the generalization accuracy of a model on future (unseen/out-of-sample) data.Methods for evaluating a model's performance are divided into 2 categories: namely, holdout and Cross-validation. Both methods use a test set (i.e data not seen by the model) to evaluate model performance

1.5.1 EVALUATION Techniques

1. Holdout

   The purpose of holdout evaluation is to test a model on different data than it was trained on. This provides an unbiased estimate of learning performance. The dataset is randomly divided into three subsets: • Training set: is a subset of the dataset (training data) used to build predictive models. We cant use all the dataset to train the model, take maximum of 60 or 70• Validation set: is a subset of the dataset used to assess the performance of the model built in the training phase. It provides a test platform for fine-tuning a model's parameters and selecting the best performing model. Not all modeling algorithms need a validation set. Take like 15 or 20• Test set (unseen data): is a subset of the dataset used to assess the likely future performance of a model. If a model fits to the training set much better than it fits the test set, overfitting is probably the cause. Take like 15 or 20The holdout approach is useful because of its speed, simplicity, and flexibility. However, this technique is often associated with high variability since differences in the training and test dataset can result in meaningful differences in the estimate of accuracy.

2. Cross-validation

   As there is never enough data to train your model, removing a part of it for validation poses a problem of underfitting. By reducing the training data, we risk losing important patterns/ trends in data set, which in turn increases error induced by bias. So, what we require is a method that provides ample data for training the model and also leaves ample data for validation. K Fold cross validation does exactly that. k-fold cross-validation is most common cross-validation technique, where the

original dataset is partitioned into k equal size subsamples, called folds. The k is a user-specified number, usually with 5 or 10 as its preferred value. This is repeated k times, such that each time, one of the k subsets is used as the test set/validation set and the other k-1 subsets are put together to form a training set. The error estimation is averaged over all k trials to get the total effectiveness of our model.

1.5.2 EVALUATION Metrics

Evaluation metrics are required to quantify model performance. The choice of evaluation metrics depends on a given machine learning task (such as classification, regression, ranking, clustering, topic modeling, among others). Some metrics, such as precision-recall, are useful for multiple tasks. Supervised learning tasks such as classification and regression constitutes a majority of machine learning applications. We will focus on metrics for these two supervised learning models.

Classification Accuracy

Accuracy is a common evaluation metric for classification problems. It's the number of correct predictions made as a ratio of all predictions made. When performing classification predictions, there's four types of outcomes that could occur.

1. True Positives

   are when you predict an observation belongs to a class and it actually does belong to that class.

2. True Negatives

   are when you predict an observation does not belong to a class and it actually does not belong to that class.

3. False Positives

   occur when you predict an observation belongs to a class when in reality it does not.

4. False Negatives

   occur when you predict an observation does not belong to a class when in fact it does.

These four outcomes are often plotted on a confusion matrix. The following confusion matrix is an example for the case of binary classification. You would generate this matrix after making predictions on your test data and then identifying each prediction as one of the four possible outcomes described above.

FIGURE 1.17. A confusion matrix of a binary classification [70]

The three main metrics used to evaluate a classification model are accuracy, precision, and recall. Where accuracy is defined as the percentage of correct predictions for the test data. It can be calculated easily by dividing the number of correct predictions by the number of total predictions.

$$accuracy = \frac{correctpredictions}{allpredictions}$$

And precision is defined as the fraction of relevant examples (true positives) among all of the examples which were predicted to belong in a certain class,

$$precision = \frac{truepositives}{truepositives + falsepositives}$$

meanwhile the recall is defined as the fraction of examples which were predicted to belong to a class with respect to all of the examples that truly belong in the class.

$$recall = \frac{truepositives}{truepositives + falseNegatives}$$

Regression Accuracy

Evaluation metrics for regression models are quite different than the above metrics we discussed for classification models because we are now predicting in a continuous range instead of a discrete number of classes. If your regression model predicts the price of a house to be 400K USD and it sells for405K USD,

that's a pretty good prediction. However, in the classification examples we were only concerned with whether or not a prediction was correct or incorrect, there was no ability to say a prediction was "pretty good". We have a different set of evaluation metrics for regression models.

1. Explained variance

   compares the variance within the expected outcomes, and compares that to the variance in the error of our model. This metric essentially represents the amount of variation in the original dataset that our model is able to explain.

$$EY(Ytrue, Ypred) = 1 - \frac{Yar(Ytrue, Ypred)}{Ytrue}$$

2. Mean squared error

   is simply defined as the average of squared differences between the predicted output and the true output. Squared error is commonly used because it is agnostic to whether the prediction was too high or too low, it just reports that the prediction was incorrect.

$$MSE(Ytrue, Ypred) = \frac{1}{Nsamples} \cdot \sum (Ytrue - Ypred)^2$$

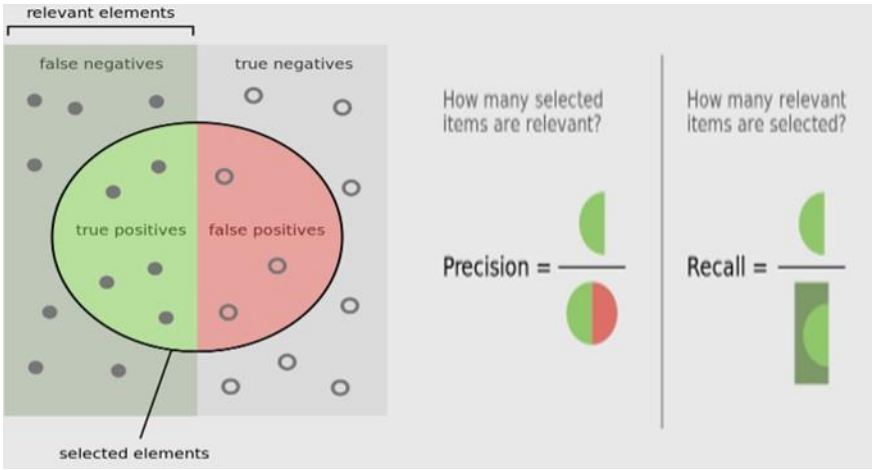

FIGURE 1.18. Classification Evaluation [71]

1.6 CONCLUSION

The conclusion in this chapter we concentrated on two parts, machine learning and neural network and how convolutional neural network works very efficiently on image recognition and classification, recurrent neural network directed more to deal with texts and we explain the solutions for RNN problems. The need for machine learning is increasing day by day. The reason behind the need for machine learning is that it is capable of doing tasks that are too complex for a person to implement directly. As a human, we have some limitations as we cannot access the huge amount of data manually, so for this, we need some computer systems and here comes the machine learning to make things easy for us. And the importance of machine learning can be easily understood by its uses cases, Currently, machine learning is used in self-driving cars, cyber fraud detection, face recognition, and friend suggestion by Facebook, etc. Various top companies such as Netflix and Amazon have build machine learning models that are using a vast amount of data to analyze the user interest and recommend product accordingly.

CHAPTER 2

RELATED Works

There have been a lot of works that dealt with text, images and 3d shapes data, as for our problem which is the generation of 3d shapes out of textual description we find it rewarding to have a look at some of the best researches treating this kind of data since the generation the 3d shapes can come in two ways, either a direct generation (shapes outputs out of text input) or generating images ,out of the text inputs, that are fed to an image to shapes generator (text-image-shape). In this chapter we take a explore some past or present state of the art approaches dealing with data in the sort of text, images or 3d shapes.

2.1 TEXT TO IMAGE

The generation of images from the regular language has numerous potential applications later on once the innovation is prepared for business applications and an amazing demonstration of Deep Learning [7]. Generative Adversarial Networks have a place with the arrangement of generative models. It implies that they can create new substances. Text is translated into picture pixels[8]. For eg: Flower with pink petals. GAN comprise of an arrangement of two contending neural organization models that compete with one another and observe, catch and duplicate the varieties inside a dataset [9]. Text to image synthesis is all about converting text descriptions into appropriate images. Nowadays, GAN models are widely used for better results [7]. We focused on two approaches used with GAN and [7] [8] [10] [11] [12] have a good explanation and demonstration to those two architectures.

2.1.1 CONDITIONAL-GANS

We take [8] as example to explain it, in their paper defined C-GANs work by inputting a one-hot class label vector as input to the generator and discriminator in addition to the randomly sampled noise vector. This results in higher training stability, more visually appealing results, as well as controllable generator outputs. The goal of Generator is to fool the Discriminator whereas the goal of

Discriminator is to identify correct data. Generator and Discriminator both compete with each other. Generator makes all the attempts to convince the Discriminator that the generated fake instances are the real samples of data and also increases the probability of mistakes whereas the Discriminator figures out the real ones. Hence, these steps are repeated many times and both the sub-models get trained much better. First, Discriminator is trained on the real data samples to verify if it can identify those samples as real [10]. Again, the Discriminator is trained on generated fake data to see if is able to discriminate between actual and fake image. Generator is also trained depending upon the results of Discriminator so it can improve itself. In addition to constructing good text embeddings, translating from text to images is highly multi-modal [7]. The term 'multi-modal' is an important one to become familiar with in Deep Learning research. This refers to the fact that there are many different images of birds with correspond to the text description "bird" [7]. Another example in speech is that there are many different accents, etc. that would result in different sounds corresponding to the text "bird". Multi-modal learning is also present in image captioning, (image-to-text). However, this is greatly facilitated due to the sequential structure of text such that the model can predict the next word conditioned on the image as well as the previously predicted words. Multi-modal learning is traditionally very difficult, but is made much easier with the advancement of GANs (Generative Adversarial Networks), this framework creates an adaptive loss function which is well-suited for multi-modal tasks such as text-to-image.

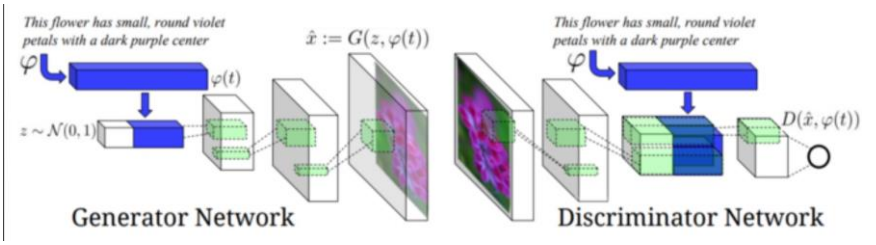

FIGURE 2.1. Reed et al. text-to-image GAN model [72]

The picture above shows the architecture Reed et al.[79] used to train this text-to-image GAN model. The most noteworthy takeaway from this diagram is the visualization of how the text embedding fits into the sequential

processing of the model. In the Generator network, the text embedding is filtered through a fully connected layer and concatenated with the random noise vector z. In this case, the text embedding is converted from a 1024x1 vector to 128x1 and concatenated with the 100x1 random noise vector z. On the side of the discriminator network, the text-embedding is also compressed

through a fully connected layer into a 128x1 vector and then reshaped into a 4x4 matrix and depth-wise concatenated with the image representation. This image representation is derived after the input image has been convolved over multiple times, reduce the spatial resolution and extracting information. This embedding strategy for the discriminator is different from the conditional-GAN model in which the embedding is concatenated into the original image matrix and then convolved over [10]. One general thing to note about the architecture diagram is to visualize how the DCGAN upsamples vectors or low-resolution images to produce high-resolution images. You can see each de-convolutional layer increases the spatial resolution of the image [12]. Additionally, the depth of the feature maps decreases per layer. Lastly, you can see how the convolutional layers in the discriminator network decreases the spatial resolution and increase the depth of the feature maps as it processes the image. An interesting thing about this training process is that it is difficult to separate loss based on the generated image not looking realistic or loss based on the generated image not matching the text description. The authors of the paper describe the training dynamics being that initially the discriminator does not pay any attention to the text embedding, since the images created by the generator do not look real at all [10]. Once G can generate images that at least pass the real vs. fake criterion, then the text embedding is factored in as well. The authors smooth out the training dynamics of this by adding pairs of real images with incorrect text descriptions which are labeled as 'fake'. The discriminator is solely focused on the binary task of real versus fake and is not separately considering the image apart from the text. This is in contrast to an approach such as AC-GAN with one-hot encoded class labels. The AC-GAN discriminator outputs real vs. fake and uses an auxiliary classifier sharing the intermediate features to classify the class label of the image. [11]

2.1.2 STACK-GAN

To generate high-resolution images with photo-realistic details, [11] propose a simple yet effective Stacked Generative Adversarial Networks. It decomposes the text-to-image generative process into two stages

Stage-I GAN: it sketches the primitive shape and basic colors of the object conditioned on the given text description, and draws the background layout from a random noise vector, yielding a low-resolution image. [12]

Stage-II GAN: it corrects defects in the low-resolution image from Stage-I and completes details of the object by reading the text description again, producing a high resolution photo-realistic image. [12]

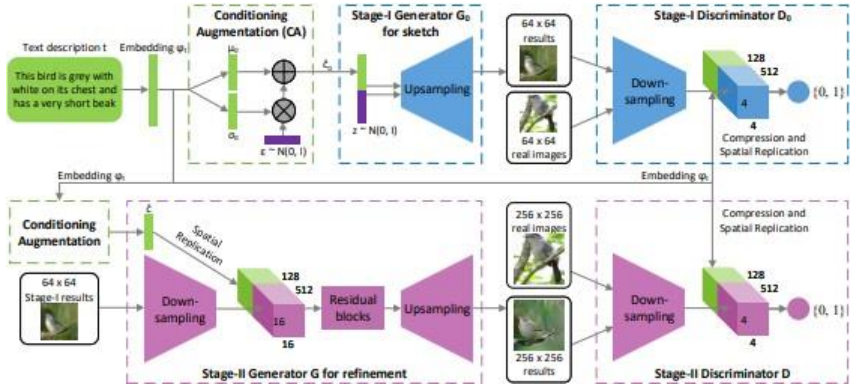

FIGURE 2.2. Stack-GAN [73]

Stage-I GAN

Instead of directly generating a high-resolution image conditioned on the text description, we simplify the task to first generate a low-resolution image with our Stage-I GAN, which focuses on drawing only rough shape and correct colors for the object. [13] Let t be the text embedding of the given description, which is generated by a pre-trained encoder in this paper [12]. The Gaussian conditioning variables c^0 for text embedding are sampled from N (μ0(t), Σ0(t)) to capture the meaning of t with variations. Conditioned on c^0 and random variable z, Stage-I GAN trains the discriminator D0 and the generator G0 by alternatively maximizing LD0 in Eq. (1) and minimizing LG0 in Eq. (2) where the real image I0 and the text description t are from the true data distribution Pdata. z is a noise vector randomly sampled from a given distribution Pz (Gaussian distribution in this paper). λ is a regularization parameter that balances the two terms in Eq. (2). We set λ = 1 for all our experiments both μ0(t) and Σ0(t) are learned jointly with the rest of the network. [12]

Stage-I GAN Architecture

For the generator G0, to obtain text conditioning variable c^0, the text embedding t is first fed into a fully connected layer to generate μ0 and σ0 (σ0 are the values in the diagonal of Σ0) for the Gaussian Distribution N (μ0(t), Σ0(t)). c^0 are then sampled from the Gaussian distribution. Our Ng dimensional conditioning vector c^0 is computed by c^0 = μ0 + σ0 (where is the element-wise multiplication, N (0, I)). Then, c^0 is concatenated with a Nz dimensional noise vector to generate a W0 $\times$H0 image by a series of up-sampling blocks. [11] For the discriminator D0, the text embedding t is first compressed to Nd dimensions using a fully-connected layer and then spatially replicated to form

a Md ×Md Nd tensor. Meanwhile, the image is fed through a series of down-sampling blocks until it has Md×Md spatial dimension. Then, the image filter map is concatenated along the channel dimension with the text tensor. The resulting tensor is further fed to a 1×1 convolutional layer to jointly learn features across the image and the text. Finally, a fully connected layer with one node is used to produce the decision score. [12]

Stage-II GAN

Low-resolution images generated by Stage-I GAN usually lack vivid object parts and might contain shape distortions. Some details in the text might also be omitted in the first stage, which is vital for generating photo-realistic images [13]. Our Stage-II GAN is built upon Stage-I GAN results to generate high-resolution images. It is conditioned on low-resolution images and also the text embedding again to correct defects in Stage-I results. The Stage-II GAN completes previously ignored text information to generate more photo-realistic details. [14]

Stage-II GAN Architecture

We design Stage-II generator as an encoder-decoder network with residual blocks. Similar to the previous stage, the text embedding t is used to generate the Ng dimensional text conditioning vector c^, which is spatially replicated to form a Mg×Mg ×Ng tensor [12]. Meanwhile, the Stage-I result s0 generated by Stage-I GAN is fed into several down-sampling blocks (i.e., encoder) until it has a spatial size of Mg×Mg. The image features and the text features are concatenated along the channel dimension. The encoded image features coupled with text features are fed into several residual blocks, which are designed to learn multi-modal representations across image and text features [14]. Finally, a series of up-sampling layers (i.e., decoder) are used to generate a W×H high-resolution image. Such a generator is able to help rectify defects in the input image while add more details to generate the realistic high-resolution image. For the discriminator, its structure is similar to that of Stage-I discriminator with only extra down-sampling blocks since the image size is larger in this stage [16]. To explicitly enforce GAN to learn better alignment between the image and the conditioning text, rather than using the vanilla discriminator, we adopt the matching-aware discriminator proposed by Reed et al.[79] for both stages. During training, the discriminator takes real images and their corresponding text descriptions as positive sample pairs, whereas negative sample pairs consist of two groups. The first is real images with mismatched text embeddings [15], while the second is synthetic images with their corresponding text embeddings.

2.2 IMAGE TO 3D SHAPES

In this chapter we take a look at some of the best works on generating 3d shapes out of images, where we will highlight the approaches used as far as shapes and images are concerned.

2.2.1 Multi-view 3D Reconstruction

3D shapes can be recovered from multiple color images or depth scans. To estimate the underlying 3D shape from multiple color images, classic SfM [16] and vSLAM [17] algorithms firstly extract and match hand-crafted geometric features [18] and then apply bundle adjustment [19] for both shape and camera motion estimation. Ji et al. [20] use "maximizing rigidity" for reconstruction, but this requires 2D point correspondences across images. Recent deep neural net based approaches tend to recover dense 3D shapes through learnt features from multiple images and achieve compelling results. To fuse the deep features from multiple images, both 3D-R2N2 [21] and LSM apply the recurrent unit GRU, resulting in the networks being permutation variant and inefficient for aggregating long sequence of images. Recent SilNet [22] [23] and DeepMVS [24] simply use max pooling to preserve the first order information of multiple images, while RayNet [25] applies average pooling to reserve the first moment information of multiple deep features. MVSNet proposes a variance-based approach to capture the second moment information for multiple feature aggregation. These pooling techniques only capture partial information, ignoring the majority of the deep features. Recent SurfaceNet [26] and SuperPixel Soup [27] can reconstruct 3D shapes from two images, but they are unable to process an arbi- trary number of images. As for multiple depth image reconstruction, the traditional volumetric fusion method [28] integrates multiple viewpoint information by averaging truncated signed distance functions (TSDF). Recent learning based OctNetFusion [29] also adopts a similar strategy to integrate multiple depth information. However, this integration might result in information loss since TSDF values are averaged [29]. PSDF [30] is recently proposed to learn a probabilistic distribution through Bayesian updating in order to fuse multiple depth images, but it is not straightforward to include the module into existing encoder-decoder networks.

2.2.2 Deep LEARNING on Sets

In contrast to traditional approaches operating on fixed dimensional vectors or matrices, deep learning tasks defined on sets usually require learning functions

to be permutation invariant and able to process an arbitrary number of elements in a set [31]. Such problems are widespread. Zaheer et al. [31] introduce general permutation invariant and equivariant models in [31], and they end up with a sum pooling for permutation invariant tasks such as population statistics estimation and point cloud classification. In the very recent CGQN [32], sum pooling is also used to aggregate an arbitrary number of orderless images for 3D scene representation. Gardner et al. [33] use average pooling to integrate an unordered deep feature set for classification task. Su et al. [34] use max pooling to fuse the deep feature set of multiple views for 3D shape recognition. Similarly, PointNet [35] also uses max pooling to aggregate the set of features learnt from point clouds for 3D classification and segmentation. In addition, the higher-order statistics based pooling approaches are widely used for 3D object recognition from multiple images

2.3 TEXT TO SHAPE

In the paper «Text2Shape: Generating Shapes from Natural Language » kevin et al [78] represent a method for generating 3d shapes from natural language, they split their work into two major tasks text to shape retrieval and text to shape generation. They first present a method for learning a joint text and shape representation space directly from natural language descriptions of 3D shape instances, By leveraging a new dataset of paired natural language descriptions and colored 3D shapes, their method extends learning by association[41] and metric learning[42] to jointly learn a text and 3D shape embedding that clusters similar shapes and descriptions, establishing implicit semantic connections followed by their text-to-shape generation framework. Unlike related work in text-to-image synthesis [12],[13] they do not rely on fine-grained category-level class labels or pre-training on large datasets. Furthermore, they train the text and shape encoding components jointly in an end-to-end fashion, associating similar points in our data both within a modality (text-to-text or shape-to-shape) and between the two modalities (text-to-shape). The retrieval task allows to evaluate the quality of the jointly learned text-shape embedding against baselines from prior work. As for the text-to shape generation task, kevin et al [78] focused on colored shape generation because most descriptions of shapes involve color or material properties. To address this task, they combined the joint embedding model with a novel conditional Wasserstein GAN framework, providing greater output quality and diversity compared to a conditional GAN formulation. Lastly, they used vector embedding arithmetic and the generator to manipulate shape attributes.[78]

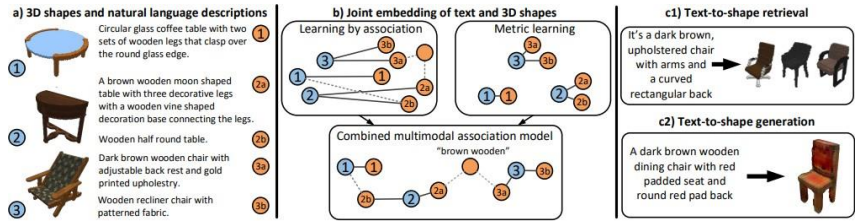

FIGURE 2.3. The approach used by kevin chen et al [78]

Experimental results on collected 75K natural language descriptions for 15K chair and table shapes in the ShapeNet[43] dataset shows that they model outperforms the baselines by a large margin for both the retrieval and generation tasks.

2.4 COMPARING THE DIFFERENT RELATED WORKS

Work	Type	Cons	Pros
Conditional GANs [10]	Text to image	1. Does not ensure the quality of generation	1. Generates realistic images from text with desirable characters
Stack GAN [12]	Text to image	1. Does not capture fine details of the image	1. improves the quality of the generated images
Multi view 3D reconstruction [20]	Image to shape	1. Generated shapes are images from the input and not new	1. Learns from one or multiple images
Deep learning on sets [31]	Image to shape	1. Shapes with low quality	1.Generates new shapes
Kevin Chen et al's work [78]	Text to shape	1. A classification problem 2. Made specifically (and only) for the shapenet chairs and tables dataset 3. the embedding task is a basic one	1. Do not rely entirely fine-grained category level annotations 2. define a metric learning loss, what improves the cross model associations
Our work	Text to shape	1. Works only with 3d objects	1. A regression problem 2. works with all the datasets of 3D shapes 3. Uses transformers and an autoencoder to compute the embeddings 4. Time and computation power efficient

2.5 CONCLUSION

In this chapter we explained 3 main related works: Text to image, Image to shape, and text to shape. We understood that the first class aimed to generate images from textual descriptions, we chose about two methods (CGAN and Stack Gan), each time we note that there are points added at the level of the approach compared to the previous one. The second part aims to transform images into shapes, we take into consideration these two approaches (Multi-view 3D Reconstruction and Deep Learning on Sets). According to the classification of the work that we made we understood that the first two parts gave birth to the third part text to shape which is the basis of our work such that the latter it uses the architecture of CWGAN. In next chapter, we will move on to the conceptual part, which will be devoted to expressing the architectures and methods used in the realization of our approach.

CHAPTER 3

Design AND conception

In the previous chapters, we have seen the different works handling data of the sort of 3d shapes, images and text, now we will walk you through our own approach and the models we chose and developed to get better results than the previews ones. First, we will explain how we generated the text embeddings using Google's BERT which is the current state of the art world wide, and then we will explain the Autoencoder we used to retrieve the shapes embeddings, to finish off by talking about the CGAN that learns the mappings between the shapes and their textual descriptions as well as generates the shape described by the text input which is the global goal of our work.

3.1 THE GLOBAL ARCHITECTURE

As shown in the previous figure, our, simple yet effective, approach consists of two main blocks the encoding and the generation. The input textual description is fed into an encoder that generates ,through BERT, its embedding, then, concatenated with some noise, will be passed to the generator that generates what it thinks is the shape associated to that description, however, to make sure the generator does its work properly we also feed the discriminator, which is another neural network that binary classifies the generated shape to a real or a fake shape in other words it tries to figure out if one shape is real and the other one is generated or are the (if the discriminator thinks the generated shape is identical to the real one we call that the discriminator being fooled by the generator and it means that the generation process was successful).the shapes are all encoded and decoded when needed to be, we do that by a pretty loss efficient autoencoder that generates the shapes embeddings and then decodes them back into the actual shapes while calculating and improving its loss function.

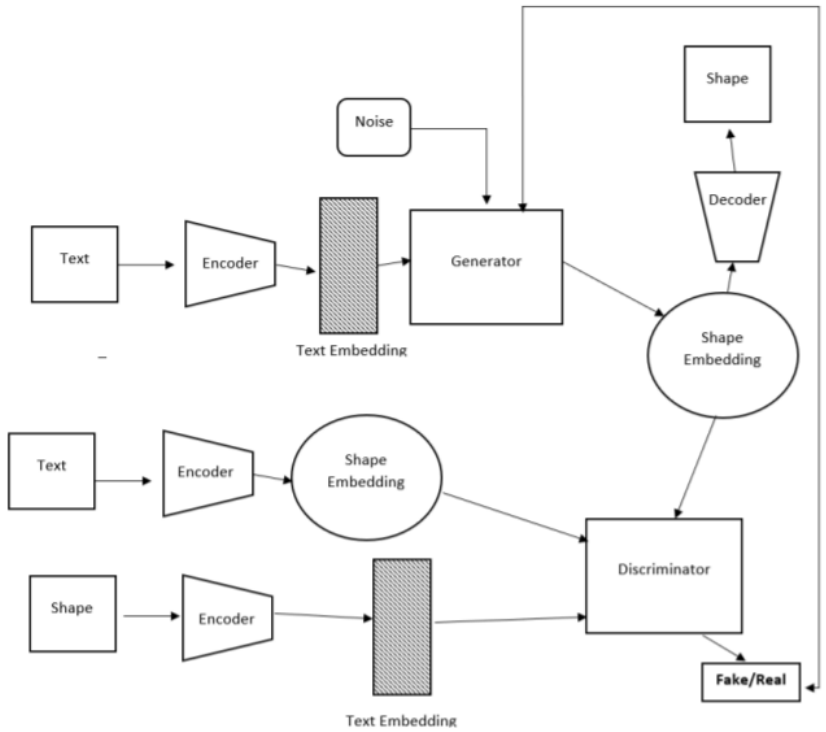

FIGURE 3.1. Our global architecture

3.2 PROCESSING THE NATURAL LANGUAGE DATA

Undoubtedly, Natural Language Processing (NLP) research has taken enormous leaps after being relatively stationary for a couple of years [21]. In this part, we will be looking at word embeddings and see how BERT can be used with word-embedding strategies to feed as input features for other models built for custom tasks to perform the state of art results. [22]

3.2.1 Word embedding

"Word embedding is the collective name for a set of language modeling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers." [23] [22] Define simply word embeddings as vector representations of a particular word. Word embedding is one of the most popular representations of document vocabulary. It is capable of capturing the context of a word in a document, semantic and syntactic similarity, relation with other words, etc. And [21] explained embeddings are low dimensional representations of a point in a higher dimensional vector space. In the same manner, word embeddings are dense vector representations of words in lower dimensional space. [22] Said that there are a few key characteristics to a set of useful word embeddings:

1. Every word has a unique word embedding (or "vector"), which is just a list of numbers for each word.

2. The word embeddings are multidimensional; typically for a good model, embeddings are between 50 and 500 in length.

3. The word embeddings are multidimensional; typically for a good model, embeddings are between 50 and 500 in length.

4.For each word, the embedding captures the "meaning" of the word.

5.Similar words end up with similar embedding values.

There are many approaches to generate word embeddings. Context-independent (Bag of Words, TF-IDF, Word2Vec, GloVe), Context-aware (ELMo, Transformer, BERT, Transformer-XL), Large model (GPT-2, XLNet, Compressive Transformer) are the main categories. We will focus on Word2Vec and Bert.

3.2.2 Common Embedding techniques

Word2Vec is one of the most popular technique to learn word embeddings using shallow neural network. It was developed [24] [14] explaind word2vec with an example, Consider the following similar sentences: Have a good day and Have a great day. They hardly have different meaning. If we construct an exhaustive vocabulary (let's call it V), it would have V = Have, a, good, great, day. Now, let us create a one-hot encoded vector for each of these words in V. Length of our one-hot encoded vector would be equal to the size of V (=5). We would have a vector of zeros except for the element at the index representing the corresponding word in the vocabulary. That particular element would be one. The encodings below would explain this better. Have = [1,0,0,0,0]; a= [0,1,0,0,0]; good= [0,0,1,0,0]; great= [0,0,0,1,0]; day= [0,0,0,0,1] If we try to visualize these encodings, we can think of a 5 dimensional space, where each word occupies one of the dimensions and has nothing to do with the rest (no projection along the other dimensions). This means 'good' and 'great' are as different as 'day' and 'have', which is not true. Our objective is to have words with similar context occupy close spatial positions. Mathematically, the cosine of the angle between such vectors should be close to 1, i.e. angle close to 0. Here comes the idea of generating distributed representations. Word2Vec is a method to construct such an embedding. It can be obtained using two methods (both involving Neural Networks): Skip Gram and Common Bag Of Words (CBOW) In the CBOW model, the distributed representations of context (or surrounding words) are combined to predict the word in the middle. While in the Skip-gram model, the distributed representation of the input word is used to predict the context. [25] Both have their own advantages and disadvantages. According to Mikolov, Skip Gram works well with small amount of data and is found to represent rare words well. On the other hand, CBOW is faster and has better representations for more frequent words. [26]

3.2.3 BERT Embedding

In 2018, the Google AI team made a revolutionary change in the field of Natural Language Processing (NLP) by introducing Bidirectional Encoder Representations from Transformers (BERT). Due to its highly pragmatic approach, and higher performance, BERT is highlighted for achieving state-of-the-art performance in many NLP tasks [11]. BERT has an advantage over models like Word2Vec because while each word has a fixed representation under Word2Vec regardless of the context within which the word appears, BERT produces word representations that are dynamically informed by the words around them [22].

For example, given two sentences:

1.I like apples.

2.I like Apple macbooks

Note that the word apple has a different semantic meaning in each sentence. Now with a contextualized language model, the embedding of the word apple would have a different vector representation which makes it even more powerful for NLP tasks [21]. the context-informed word embeddings capture other forms of information that result in more accurate feature representations, which in turn results in better model performance [22]

BERT Model

Two primary models were created by BERT developers:

1. The BASE:Number of transformer blocks (L): 12, Hidden layer size (H): 768 and Attention heads(A): 12

2. The LARGE:Number of transformer blocks (L): 24, Hidden layer size (H): 1024 and Attention heads(A): 16

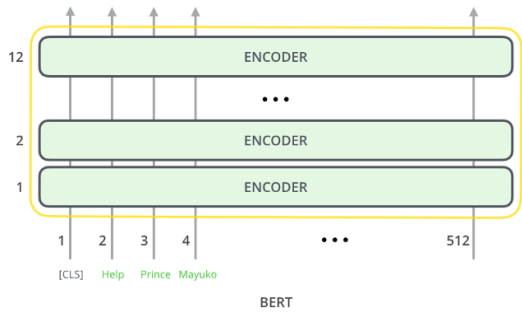

FIGURE 3.2. a graphical representation of the base BERT model [74]

It may seem simple but each encoder block encapsulates a more sophisticated model architecture. At this point, to make things more clear it is important to understand the special tokens that BERT authors used for fine-tuning and specific task training. [11] These are the following:

1. CLS: The first token of every sequence. A classification token which is normally used in conjunction with a softmax layer for classification tasks. For anything else, it can be safely ignored.

2. SEP: A sequence delimiter token which was used at pre-training for sequence-pair tasks (i.e. Next sentence prediction). Must be used when sequence pair tasks are required. When a single sequence is used it is just appended at the end.

3. MASK: Token used for masked words. Only used for pre-training.

Moving on, the input format that BERT expects is illustrated below:

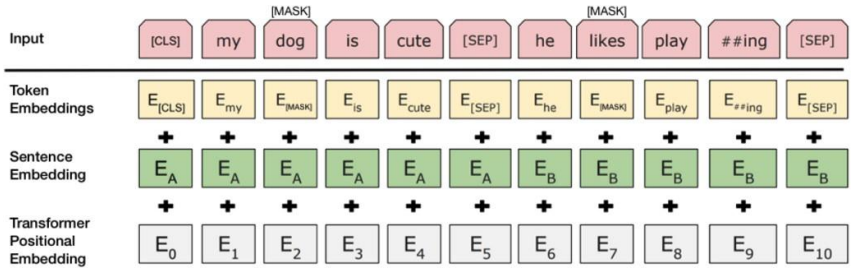

FIGURE 3.3. an example of inputting a sentence into BERT [74]

The input layer is simply the vector of the sequence tokens along with the special tokens. BERT use WordPiece for tokenization which in effect, splits token like "playing" to "play" and "ing". This is mainly to cover a wider spectrum of Out-Of-Vocabulary (OOV) words [22]. Token embeddings are the vocabulary IDs for each of the tokens, and a Sentence Embedding is just a numeric class to distinguish between sentence A and B, while Transformer positional embeddings indicate the position of each word in the sequence.

3.2.4 GENERATING our text embeddings:

BERT might be known for its words embedding capabilities but the sentence encoding models also are easily on of the best pre-trained models to be used to generate sentences embeddings, for instance, all-MiniLM-L6-v2 [**](that we use to compute the text embeddings) is a sentence transformer model that maps sentences paragraphs to a 384 dimensional dense vector space and can be used for tasks like clustering or semantic search. it is intended to be used as a sentence and short paragraph encoder. Given an input text, it outputs a vector which captures the semantic information. The sentence vector may be used for information retrieval, clustering or sentence similarity tasks. By default, input text longer than 256 words pieces is truncated. We chose this particular model because it fits perfectly our captions dataset. in the figure bellow its main characteristics.

all-MiniLM-L6-v2

Description:	All-round model tuned for many use-cases. Trained on a large and diverse dataset of over 1 billion training pairs.
Base Model:	nreimers/MiniLM-L6-H384-uncased
Max Sequence Length:	256
Dimensions:	384
Normalized Embeddings:	true
Suitable Score Functions:	dot-product (`util.dot_score`), cosine-similarity (`util.cos_sim`), euclidean distance
Size:	80 MB
Pooling:	Mean Pooling
Training Data:	1B+ training pairs. For details, see model card.
Model Card:	https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

FIGURE 3.4. all-MiniLM-L6-v2 model information

We simply download the pre trained model and then call the 'encode()' function on the captions dataset and it generates the each caption's embedding, the process might take some time but nothing fancy since out of the 6 sentence embedding BERT models with the highest performance this model is the fastest (on a V100 GPU it encodes 14200 sentence per second [27]).

3.3 PROCESSING THE 3D SHAPES

3d shapes are such high dimensional vectors and that makes them costly ,if not impossible, to be used in machine learning tasks, therefore we are entitled to generate embeddings that captures the our shapes' main characteristics while ,at the same time, easier to do machine learning on large inputs of them, this only be done by focusing on properties of high importance, properties that distinguish each shape form another (color, material, finishing.) and ignoring what is repetitive, irrelevant and non schematic(to escape the overfitting problem). We propose an Autoencoder that encodes and decodes back the 3d shapes multiple times improving its loss function every time, this way we can enjoy representative shapes embeddings. An Autoencoder has two main blocks the encoder and the decoder, the encoder encodes shapes into the wanted embeddings while the decoder gets that embedding and decodes it to get back the original shape. Their architecture is simple, the encoder is made out of two conv3d layer (3 dimensional filter) followed both by a rectifier linear unit activation function (that outputs the the value if positive otherwise it will output 0) the first one uses a kernel of (32, 32, 32) and a stride of (2,1,1) with padding=(4, 2, 0) while the second one convolves through the output of the first layer with a kernel of size (5, 5, 1) with no stride value nor padding (both set to 1). the decoder on the other hands applies a 3D transposed convolution operator over the embeddings generated by the encoder, that be done by two transposed convolutional layer layers with the same parameters in the inverse order (the first layer uses a kernel of (5,5,1) and no padding nor stride ad the first one kernel=(32, 32, 32), stride=(2, 1, 1), padding=(4, 2, 0))

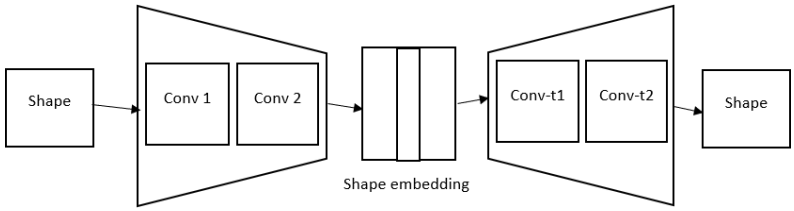

FIGURE 3.5. Our autoencoder architecture

We improve the encoding and decoding of the model by applying the mean squared error loss between the generated shape of the decoder and the shape

inputted first into the encoder and updating the weights of the layers during the back propagation process. We train the model for 100 epochs using a batch of 4 shapes at a time (for performance reasons).

3.4 THE GENERATION TASK

We chose GANs as a model for this task, generative adversarial networks are a type of neural networks that uses two blocks called the generator and the discriminator to understand as well as generate complex data using mathematical operations, which is very helpful since we are trying to generate data out of another, completely different, type (generating 3d shapes out of textual descriptions), In the normal case the generator generates things out of random noise (random vectors) but what we are trying to do here is a little bit different, therefore we will make use of a condition that we inject into the generator to make sure it always stays close to the textual description inputted (serves as creating a certain pattern of generation that's why it is called the conditional generative adversarial network). After getting the shapes and captions embeddings (using respectively our autoencoder and Google's BERT model) we now join the shapes with their textual descriptions what gives us a full labeled dataset of low dimensional train ready shapes embeddings associated with their respective textual description also encoded, now we move to the generation task.As mentioned before our CGAN has two main blocks: the generator and the discriminator, we first feed the embedded textual descriptions ,that are a latent vector of size 384, concatenated with a random nosy vector into the generator that will output a 3d shape of the dimension (4,3,3,3) and then fed , alongside with the embedding of the original shape that is described here, straight into the discriminator for it to classify the generated shape into a real or fake (figures out if it was a generated one or not) and then sends the result back to the generator therefore it learns if it did a good job or not and keeps repeating through many epochs improving every time (most of the time is more accurate), if the discriminator thinks that the generated shape is a completely identical to the real one then we can say that the generation part was a success in fact if the versions are identical in a way that fools the discriminator it is enough and gets the job done.The generator and the discriminator, as you might have guessed, aren't anywhere near similar in their architecture, since they do completely different tasks, the generator is a sequential model of 5 linear layers that are, mathematically, designed to calculate the linear equation Ax = b where x is input, b is output, A is weight and applies the linear transformation to the given input into another size, and that what allows us to move from a latent vector of size 384 to a shape of (4,32,32,32) at the end, the first four layers are followed by a 1 dimensional batch normalizing layer, except for the first one, and each of the four uses a leaky rectified linear unit activation function. The last one outputs the shape embedding (of size (4,3,3,3)) and it uses the Tanh activation function. Although we intended to generate shapes, we thought it would be better to directly generate the shapes embeddings, we tried them both but the

embedding generation was way better time and performance wise,therefore the generation of shapes embeddings gets the latent vector of size 384 and outputs the embedding of size (4,3,3,3).

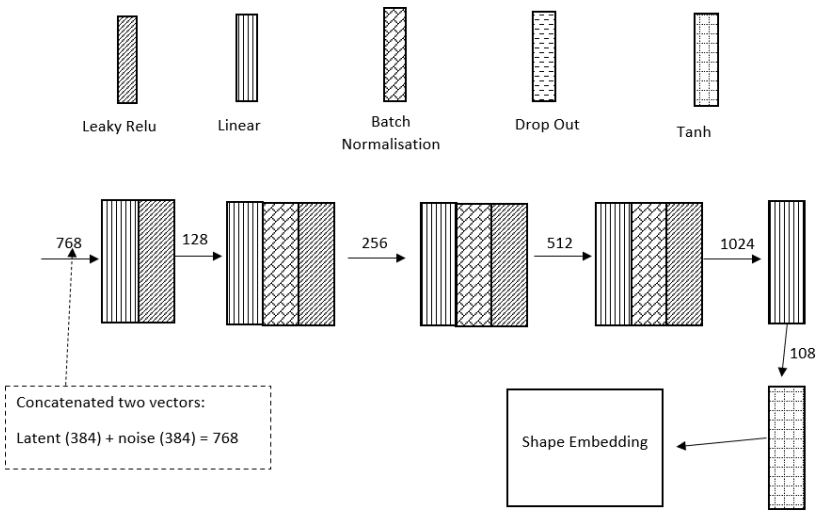

FIGURE 3.6. Our CGAN's generator architecture

On the other hand , in the discriminator we find 4 linear layers, the first one has as an input feature 492 and outputs a feature map of 512that is passed through a leaky rectified linear units and then through two linear layers with a dropout function with a probability of 0.4 what that does is it keeps the same size of the vector but zeroes some of its values with the probability of 40 percent an element is zeroed, at the end the mostly zeroed vector gets fed to a linear layer that outputs one value which is if it is a fake or a real shape, and the classification accuracy gets improved in the backpropagation process after every epoch and this way it motivates the generator to step up its game and generate challenging shapes for the discriminator to discriminate between them and the real ones, that's how we get the final generator that can generate close to real shapes out of nothing but textual descriptions.

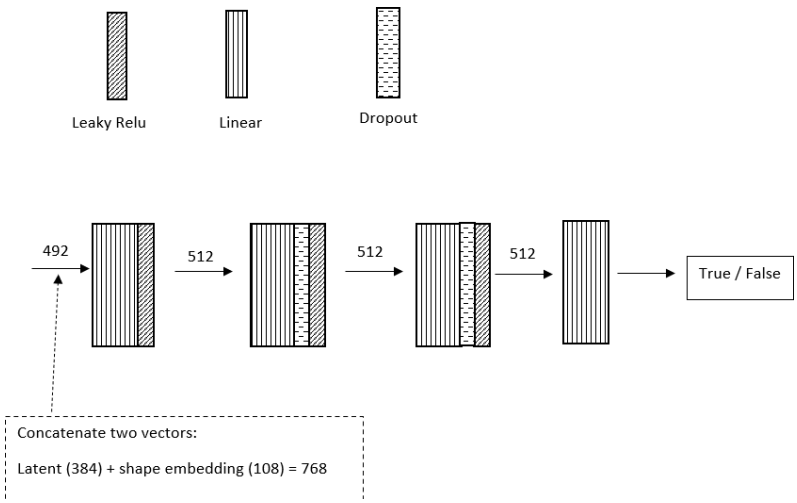

FIGURE 3.7. Our CGAN's discriminator architecture

3.5 CONCLUSION

The chapter we just walked you through is the most important one in thesis since it is the one in which we explained in details what are we doing as well as how we are doing it. We can divide it into three important parts the text embeddings extraction where we explained how we used Google's BERT to get the embeddings of our dataset captions, and then in the second part we talked about our Autoencoder that generates the shapes embeddings and detailed its structure and how it works, in the last important part we tackled the CGAN that we used in our generation task. In the next and last chapter, we will talk about the tools we used and evaluate our work.

CHAPTER 4

The IMPLEMENTATION of our work

In the previews chapter we discussed how our models are made and their architecture in this chapter we will discuss the process of building them, starting with the tools we used moving to parameters we build them upon and finishing with a brief overview evaluation .

4.1 USED TOOLS:

in this section we will show you what we used to make this whole thing work.

4.1.1 Google COLLABORATORY

Google Colaboratory (also known as Colab) is a free Jupyter notebook environment that runs in the cloud and stores its notebooks on Google Drive., it allows you to write and execute Python in your browser, with zero configuration required, free access to GPUs and easy sharing, also since it uses google drive you can easily read data from drive in Colab and use it which is one of the best features out there for developers world wide.

4.1.2 Python

Python is a programming language that has become a staple in data science, allowing data analysts and other professionals to use the language to conduct complex statistical calculations, create data visualizations, build machine learning algorithms, manipulate and analyze data, and complete other data-related tasks. Python can build a wide range of different data visualizations, like line and bar graphs, pie charts, histograms, and 3D plots. Python also has a number of libraries that enable coders to write programs for data analysis and machine learning more quickly and efficiently, like TensorFlow, Pytorch and Keras.

4.1.3 Frontend we development tools

For our interface we used HTML, CSS and JS, in the interface there is a text area so can the user put a small description describe the shape he wants, and click generate the description treated in our backend and generate the shape.

4.1.4 F3D

F3D is a desktop program made by c++ and can display 3D shapes in NRRD format (the main format we are using in our work). After our model generates the shapes we drag and drop the generated NRRD file in the program and it displays it.

4.1.5 VISUAL studio code

Visual Studio Code is a streamlined code editor with support for development operations like debugging, task running, and version control. It aims to provide just the tools a developer needs for a quick code-build-debug cycle and leaves more complex workflows to fuller featured IDEs

4.1.6 NumPy

NumPy is an open source project aiming to enable numerical computing with Python. It was created in 2005, building on the early work of the Numeric and Numarray libraries[49]

4.1.7 NRRD

Nrrd ("nearly raw raster data") is a library and file format for the representation and processing of n-dimensional raster data. It was developed by Gordon Kindlmann to support scientific visualization and image processing applications.it can be used, accessed and modified through python's library Pynrrd.

4.1.8 Bert

Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based machine learning technique for natural language processing (NLP) pre-training developed by Google. it is designed to pre-train deep

bidirectional representations from unlabeled text by jointly conditioning on both left and right context. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of NLP tasks.

4.1.9 Pytorch

PyTorch is an open source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing, primarily developed by Facebook's AI Research lab (FAIR). Although the Python interface is more polished and the primary focus of development, PyTorch also has a C++ interface.

4.2 DATASETS

ShapeNet is a large, information-rich repository of 3D models. It contains models spanning a multitude of semantic categories. Unlike previous 3D model repositories, it provides extensive sets of annotations for every model and links between models in the repository and other multimedia data outside the repository. Like ImageNet, ShapeNet provides a view of the contained data in a hierarchical categorization according to WordNet synsets, Unlike other model repositories, ShapeNet also provides a rich set of annotations for each shape and correspondences between shapes. The annotations include geometric attributes such as upright and front orientation vectors, parts and keypoints, shape symmetries , and scale of object in real world units. These attributes provide valuable resources for processing, understanding and visualizing 3D shapes in a way that is aware of the semantics of the shape.

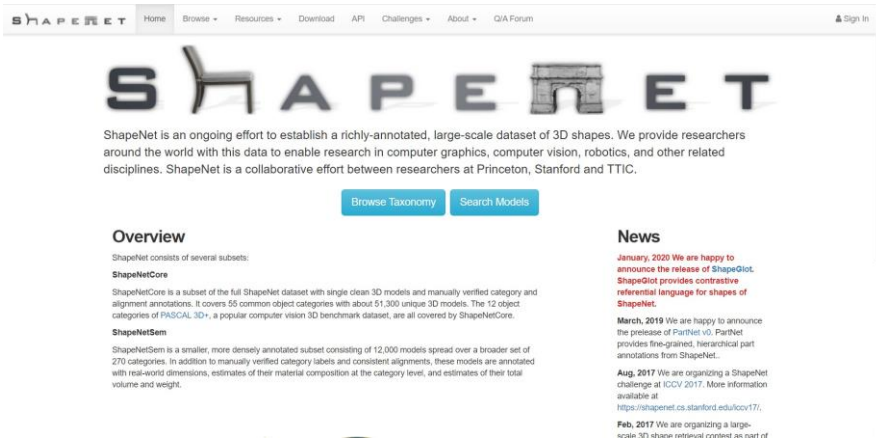

FIGURE 4.1. The shapenet project home page [76]

To create a realistic dataset with real 3D objects and natural language descriptions, we use the ShapeNet table and chair object categories (with 8,447 and 6,591 instances, respectively). These 3D shapes were created by human designers to accurately represent real objects.

FIGURE 4.2. Example form the shapenet tables and chairs dataset [76]

We choose the table and chair categories because they contain many instances with fine-grained attribute variations in geometry, color and material. We augment this shape dataset with 75,344 natural language descriptions (5 descriptions on average per shape) provided by people on the Amazon Mechanical Turk crowdsourcing platform augmented with natural language descriptions, and a controlled, procedurally generated dataset of 3D geometric primitives. This large-scale dataset provides many challenging natural language descriptions paired with realistic 3D shapes.

FIGURE 4.3. paired sahpes and descriptions from our dataset [76]

To enable systematic quantitative evaluation of our model, we use a dataset of 3D geometric primitives with corresponding text descriptions. This data

was generated by voxelizing 6 types of primitives (cuboids, ellipsoids, cylinders, cones, pyramids, and tori) in 14 color variations and 9 size variations. The color and size variations are subjected to random perturbations generating 10 samples from each of 756 possible primitive configurations, thus creating 7560 voxelized shapes. They, then, created corresponding text descriptions with a template-based approach that fills in attribute words for shape, size, and color in several orderings to produce sentences such as "a large red cylinder is narrow and tall". In total, we generate 192,602 descriptions, for an average of about 255 descriptions per primitive configuration. Such synthetic text does not match natural language but it does allow for an easy benchmark with a clear mapping to the attributes of each primitive shape.

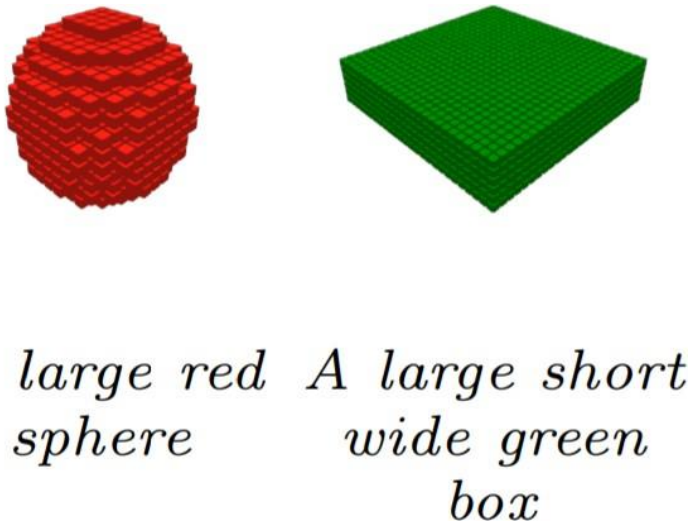

FIGURE 4.4. example from the primitives dataset [76]

4.3 OUR INTERFACE

we used a web interface for the users to interact and use our deep learning models, is a way that our from end is a simple application that has two pages one that has the input field where a user can enter its query which is in our case a textual description, that text is getting embedded and then fed straight into the generator which will generate the shape that was described textually by the user's query and send a tensor an nrrd file, that we drag and drop in the F3D interface which will visualize it as a 3d file. Below are screen shots of the whole application.

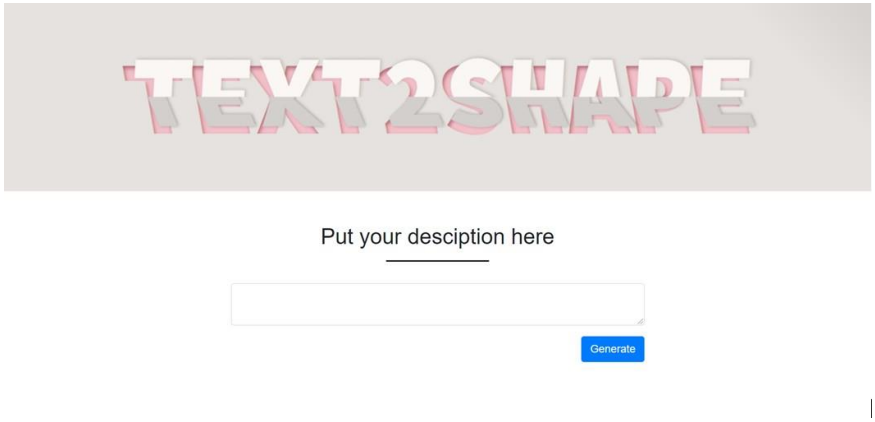

FIGURE 4.5. a screenshot of the input page of our application

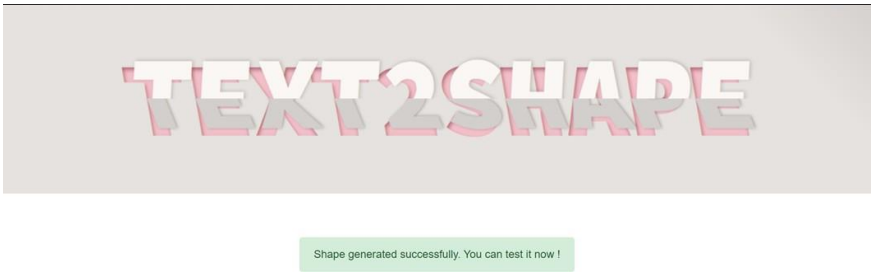

FIGURE 4.6. a screenshot of the successful generation page of our application

FIGURE 4.7. some samples generated with our model

4.4 EXPERIMENTS

in this section we will take a look at the different models that we used in our work and the configuration we used to get the best out of them (in terms of results performance and computational time).

4.4.1 Autoencoder:

We used the autoencoder to learn the shape embeddings, and it did a great job, bellow the hyper parameters we used to train the model. The number of epochs : 100 The batch size : 8 The number of worker : 4, what that means is that we will be using 4 cores to fetch and use data at a time. We also used the adam optimizer which is one of the best and widely used optimizers it computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients and speaking of the learning rate we used a learning rate of 1e-3. As for the loss we used the mean squared error loss, nothing fancy but gets the job done. Bellow is the chart of how the autoencoder learns and improves during the 100 epochs of training.

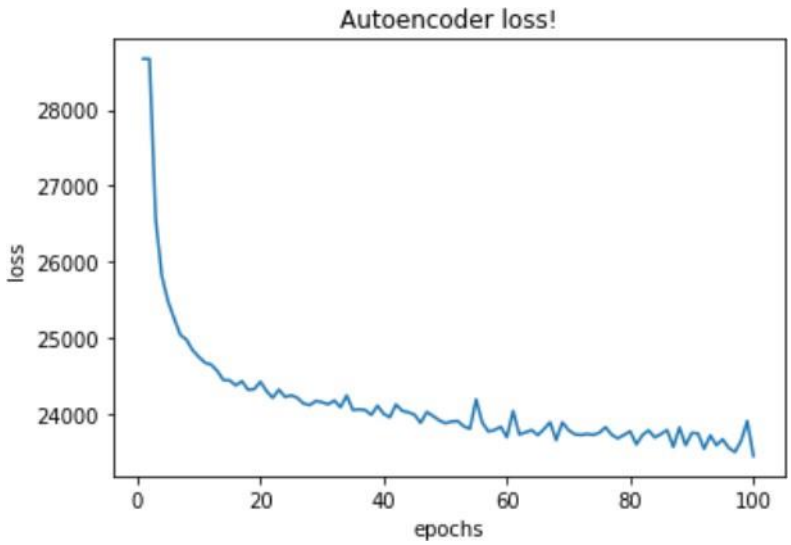

FIGURE 4.8. the loss function of the autoencoder

4.4.2 CGAN

In the generative adversarial network part, which is the global neural network of this work, the generator gets an encoded caption concatenated with a noise vector and generates a 3d shape, but we tried two different approaches, in the first one the generation outputs a 3d shape that is fed into the discriminator (with a latent vector) in the other approach we do not work with shapes but with only embeddings which we see is better. they might be different approaches but the hyper parameters are the same which they are :

The number of epochs : 200

The batch size : 32

The number of worker : 4

The learning rate : 0.0002

The latent vectors are of size 384 ,

And the shape of the shapes' embeddings is (4,3,3,3). The loss is the same Mean Squared Error.and below are the charts of all the models' losses per epoch.

CGAN with shapes:

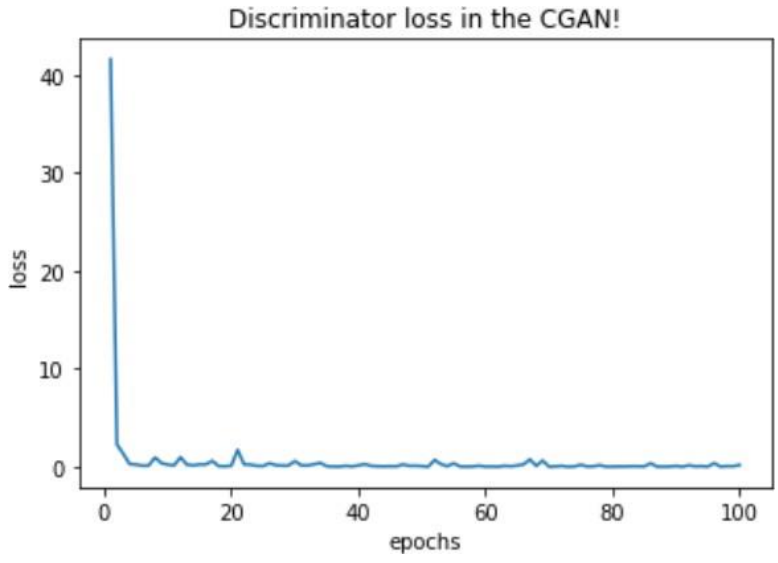

FIGURE 4.9. the loss function of the discriminator

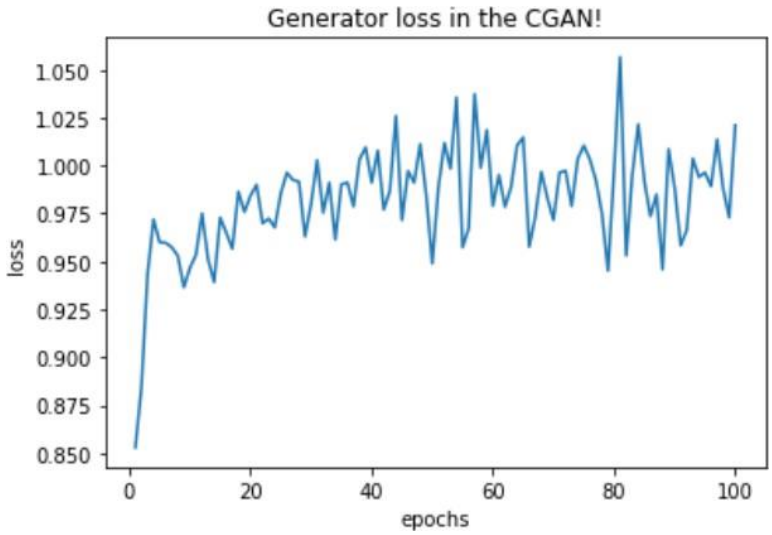

FIGURE 4.10. the loss function of the generator

CGAN with shapes embeddings:

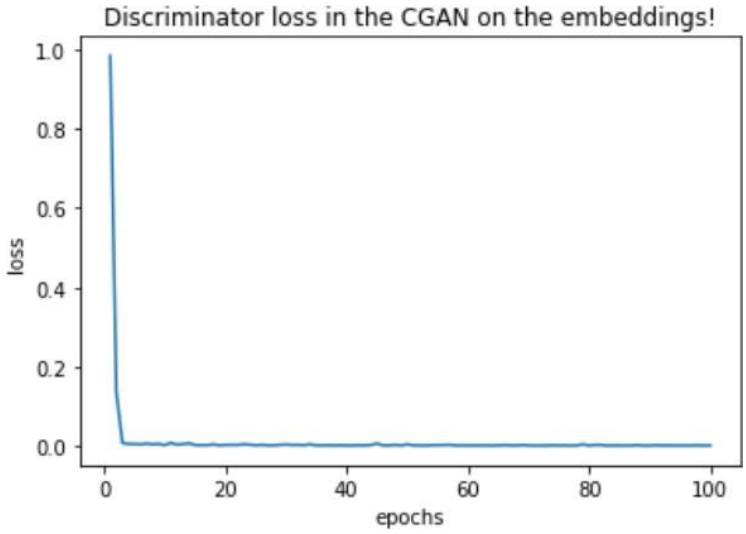

FIGURE 4.11. the loss function of the discriminator

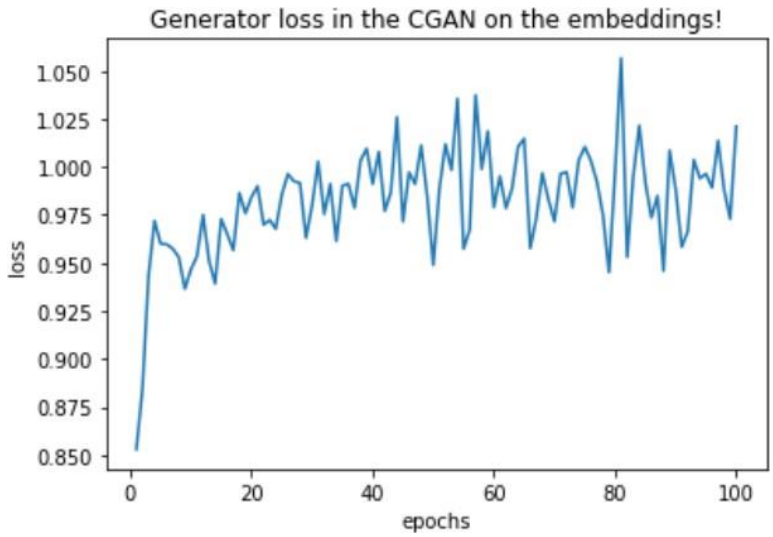

FIGURE 4.12. the loss function of the generator

4.4.3 COMPARATIVE study :

The generator using the original shapes does a better job than the one using the embeddings, not by a big margin but still can be helpful, meanwhile the discriminator using the shape embeddings outperforms the other one, again by a slight margin. And bellow is both their charts grouped together so we can see the difference clearly:

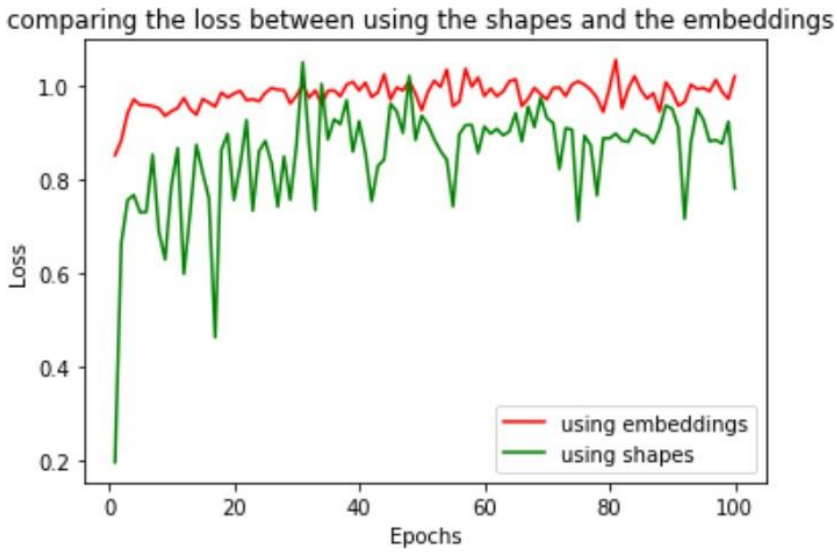

FIGURE 4.13. comparative study of the generator

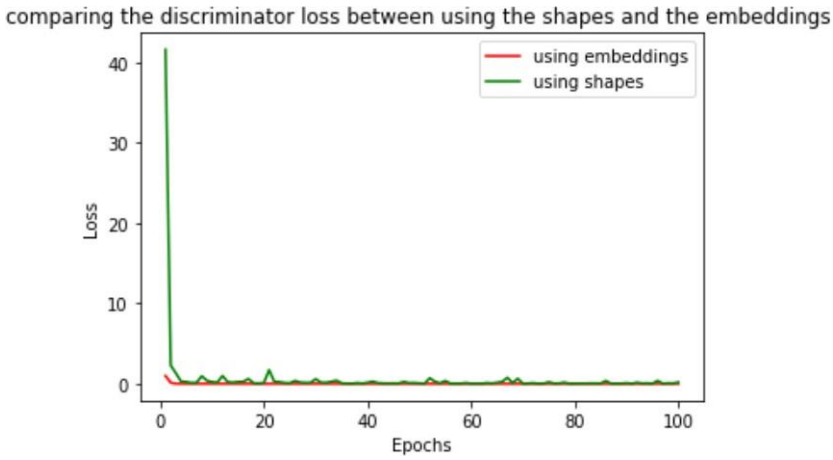

FIGURE 4.14. comparative study of the discriminator

4.4.4 Discussing the results:

We mentioned before that the generator works better with the original shapes and the discriminator with the embeddings which we think is due to the difference in size between the shape and its embedding, the shapes are so big in size

which means that are more ways for the generator to fool the discriminator and the opposite in the case of the discriminator the embeddings are too small for the generator to play with them in other words, the classification of a small amount of data is easier than a big one. But in practice we need the generator to perform better, although it is a balance we need to find between the generator and the discriminator it's the generator that is going to generate the shapes out of the captions for us. now in real practice, considering the cost of using the shapes which , in our case, took 4+ hours per epoch comparing with the embedding that only did the job in minutes and the minimum hardware requirements to do both, going with a well generated embeddings is a no brainer.

4.5 CONCLUSION:

We divided this last chapter of our work into two parts where we showed you what tools we used to get things done in the first part and then evaluated the work we did in the second part while discussing the results of the experiments we did.

General Conclusion

In this final step of this milestone project, we will take a look back at what we were doing from the beginning. We used shapenet's tables and chairs dataset of 3d chairs and tables with their textual descriptions, also the primitives which is a similar one but has only the base primitive forms (a pyramid, a ball), and then generated the caption's embeddings using Google's Bert which is the current state of the art in all the NLP related tasks. We also used google colab to design and implement an autoencoder that extracts the shapes embeddings because we thought that using shapes embeddings instead of the actual shapes would be such an optimistic way of handling the time and performance issues when dealing with data as complex as 3d objects given that the results aren't that bad at all. Then we created the generator that will be generating for us shapes given a textual description, we went for a conditional generative adversarial network because we are after a generation task that demands respect a condition (the generated shape should be generated in a way that fits the text describing it) which is only possible with a CGAN, we trained the generator and the discriminator in both the 3D shapes and their embeddings and then discussed the results of both. We learned a lot of things participating in this huge project starting with putting our hands on a real dataset to building solutions to real-world problems and finally implementing what we learned throughout our university program. Although we did end up getting some pleasing results our work is not by any means perfect, in fact we could use a transformer to extract the shape embeddings, since nothing beats a transformer when it comes to the embeddings and it easily beats our autoencoder. Also using a more realistic dataset would be help so much in increasing the quality of the generated shapes an example of that would be using shapenet's (128,128,128) shapes dataset which will allow us to generate shapes of the same size that look cleaner and more feature-rich. We focused on only chairs and tables, but the text to shape task has more to it than that, in fact there is no end to the things that could b generated out of textual descriptions, and it is a no brainer that the future generations will be investing in this new yet rapidly thriving technology, imaging being able to get a 3d shape of whatever you can describe that would open new horizons to the whole human race,

things that we needed years just to design can be put through words the most effective way human beings found ,after hundreds of decades of research, to communicate, learn and teach, it would just be enourmous.

BIBLIOGRAPHY

[1] Javatpoint supervised machine learning https://www.javatpoint.com/ supervised-machine-learning

[2] Hastie t., tibshirani r., friedman j. (2009) unsupervised learning. in: The elements of statistical learning. springer series in statistics. springer, new york, ny. https://doi.org/10.1007/978- 0-387-84858-714

[3] Zhu, xiaojin (jerry) university of wisconsin-madison semi- supervised learning literature https://minds.wisconsin.edu

[4] F. q. lauzon, an introduction to deep learning, 2012 11th international conference on infor- mation science, signal processing and their applications (isspa), 2012, pp. 1438-1439, doi: 10.1109/isspa.2012.6310529

[5] B. YEGNANARAYANA. Artificial neural networks. (). ISBN 8120312538, 9788120312531

[6] Léon Bottou Martin Arjovsky, Soumith Chintala. Wasserstein gan. https://arxiv.org/abs/1701.07875.

[7] «Text to Image » to: Connor Shorten

[8] «Text to Image using Deep Learning » Paper IJERTV10IS040132 to Akanksha Singh, Sonam Anekar, Ritika Shenoy, Sainath Patil. Published 21-04-2021

[9] «Text-to-Image Generation» https://paperswithcode.com

[10] «Text-to-Image Synthesis » Nikunj Gupta published 11-01-2019

[11] «StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks » Rutgers University, Lehigh University, The Chinese University of Hong Kong, Baidu Research

[12] «Text to Image Synthesis Using Generative Adversarial Networks» to: Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, Dimitris Metaxas

[13] Text to Photo-Realistic Image Synthesis » to: Rajat Garg published 18-02-2019

[14] « StackGAN: Text to Photo-Realistic Image Synthesis » to: Han Zhang published in 2017

[15] « Stacked generative adversarial networks for image compositing » Bing Yu, Youdong Ding, Zhifeng Xie & Dongjin Huang published in 2021

[16] «How to Convert Text to Images » to: Siraj Raval on https://youtube.com

[17] Cadena C, Carlone L, Carrillo H, Latif Y, Scaramuzza D, Neira J, Reid ID, Leonard JJ (2016) Past, Present, and Future of Simultaneous Localization and Map- ping

[18] Hartley R, Zisserman A (2004) Multiple View Geometry in Computer Vision. Cambridge University Press

[19] Triggs B, McLauchlan PF, Hartley RI, Fitzgibbon AW (1999) Bundle Adjustment - A Modern Synthesis. International Workshop on Vision Algorithms

[20] Ji P, Li H, Dai Y, Reid I (2017b) "Maximizing rigid- ity" Revisited: a Convex Programming Approach for Generic 3D Shape Reconstruction from Multiple Per- spective Views. IEEE International Conference on Computer Vision pp 929–937

[21] Choy CB, Xu D, Gwak J, Chen K, Savarese S (2016) 3D- R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction. European Conference on Computer Vision

[22] Wiles O, Zisserman A (2017) SilNet : Single- and Multi- View Reconstruction by Learning from Silhouettes. British Machine Vision Conference

[23] Wiles O, Zisserman A (2018) Learning to Predict 3D Surfaces of Sculptures from Single and Multiple Views. International Journal of Computer Vision pp 1–21

[24] Huang PH, Matzen K, Kopf J, Ahuja N, Huang JB (2018) DeepMVS: Learning Multi-view Stereopsis. IEEE Conference on Computer Vision and Pattern Recognition pp 2821–2830

[25] Paschalidou D, Ulusoy AO, Schmitt C, Van Gool L, Geiger A (2018) RayNet: Learning Volumetric 3D Reconstruction with Ray Potentials. IEEE Conference on Computer Vision and Pattern Recognition 38973906

[26] Ji M, Gall J, Zheng H, Liu Y, Fang L (2017a) Sur-faceNet: An End-to-end 3D Neural Network for Mul- tiview Stereopsis. IEEE International Conference on Computer Vision pp 2326–2334

[27] Kumar S, Dai Y, Li H (2017) Monocular Dense 3D Reconstruction of a Complex Dynamic Scene from Two Perspective Frames. IEEE International Conference on Computer Vision pp 4649–4657

[28] Curless B, Levoy M (1996) A Volumetric Method for Building Complex Models from Range Images. Conference on Computer Graphics and Interactive Techniques pp 303–312

[29] Riegler G, Ulusoy AO, Bischof H, Geiger A (2017) Oct ,NetFusion: Learning Depth Fusion from Data. International Conference on 3D Vision pp 57–66

[30] Dong W, Wang Q, Wang X, Zha H (2018) PSDF Fusion:Probabilistic Signed Distance Function for On-the-fly 3D Data Fusion and Scene Reconstruction. European Conference on Computer Vision pp 714–730

[31] Zaheer M, Kottur S, Ravanbakhsh S, Poczos B, Salakhutdinov R, Smola A (2017) Deep Sets. International Conference on Neural Information Processing Systems

[32] Eslami SA, Rezende DJ, Besse F, Viola F, Morcos AS, Garnelo M, Ruderman A, Rusu AA, Danihelka I, Gregor K, Reichert DP, Buesing L, Weber T, Vinyals O, Rosenbaum D, Rabinowitz N, King H, Hillier C, Botvinick M, Wierstra D, Kavukcuoglu K, Hassabis D (2018) Neural scene representation and rendering. Science 360(6394):1204–1210

[33] Gardner A, Kanno J, Duncan CA, Selmic RR (2017) Classifying Unordered Feature Sets with Convolutional Deep Averaging Networks. ArXiv 1709.03019

[34] Su H, Maji S, Kalogerakis E, Learned-Miller E (2015) Multi-view Convolutional Neural Networks for 3D Shape Recognition. IEEE International Conference on Computer Vision pp 945–953

[35] Qi CR, Su H, Mo K, Guibas LJ (2017) PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. IEEE Conference on Computer Vision and Pattern Recognition pp 652–660

[36] Javatpoint Supervised ML, https://www.javatpoint.com/supervised-machine-learning

[37] Hastie T., Tibshirani R., Friedman J. (2009) Unsupervised Learning. In: The Elements of Statistical Learning. Springer Series in Statistics. Springer, New York, NY. https://doi.org/10.1007/978-0-387-84858-7_14

[38] Zhu, Xiaojin University of Wisconsin Madison https://minds.wisconsin.edu/bitstream/handle/1793/60444/TR1530.pdf

[39] F. Q. Lauzon, An introduction to deep learning,2012 11th International Conference on Information Science, Signal Processing and their Applications (ISSPA), 2012, pp. 1438-1439, doi: 10.1109/ISSPA.2012.6310529.

[40] Artificial Neural Networks to B. YEGNANARAYANA

[41] Haeusser, Mordvintsev, Cremers: Learning. arXiv preprint arXiv:1706.00909 (2017)

[42] Song, H.O., Xiang, Y., Jegelka, S., Savarese, S.: Deep metric learning via lifted structured feature embedding. In: Computer Vision and Pattern Recognition (CVPR). (2016)

[43] Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M.Song, S., Su, H., Xiao, J., Yi, L., Yu, F.: ShapeNet: An information-rich 3D model repository. Technical Report arXiv:1512.03012 [cs. GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago (2015)

[44] Wasserstein GAN to Martin Arjovsky, Soumith Chintala, Léon Bottou https://arxiv.org/abs/1701.07875

[45] https://colab.research.google.com/

[46] https://www.python.org/

[47] https://f3d-app.github.io/f3d/

[48] https://code.visualstudio.com/

[49] https://numpy.org/

[50] https://pypi.org/project/pynrrd/

[51] https://blog.google/bert/

[52] https://pytorch.org/

[53] https://shapenet.org/

[54] https://www.javatpoint.com/supervised-machine-learning
[55] https://www.javatpoint.com/unsupervised-machine-learning
[56] https://fr.mathworks.com/discovery/reinforcement-learning.html
[57] https://www.technologies-ebusiness.com/enjeux-et-tendances/le-deep-learning-pas-a-pas
[58] https://www.geogebra.org/m/DK7UF2rB
[59] https://www.researchgate.net/figure/The-most-common-nonlinear-activation-functions_fig1_309775740
[60] https://www.researchgate.net/figure/An-example-of-a-Supervised-Learning-classification-of-cats-and-dogs-and-b_fig1_328576527
[61] https://anhreynolds.com/blogs/cnn.html
[62] https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7
[63] https://analyticsindiamag.com/comprehensive-guide-to-different-pooling-layers-in-deep-learning/
[64] https://medium.com/swlh/fully-connected-vs-convolutional-neural-networks-813ca7bc6ee5
[65 https://fr.wikipedia.org/wiki/reseau_de_neurones_recurrents
[66] https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/
[67] https://blog.keras.io/building-autoencoders-in-keras.html
[68] https://developers.google.com/machine-learning/gan/gan_structure
[69] https://medium.com/@ma.bagheri/a-tutorial-on-conditional-generative-adversarial-nets-keras-implementation-694dcafa6282
[70] https://www.researchgate.net/figure/Confusion-Matrix-for-binary-classification_fig5_346390613
[71] https://www.nature.com/articles/nmeth.3945
[72] https://towardsdatascience.com/text-to-image-a3b201b003ae
[73] https://www.arabicprogrammer.com/article/12341017339/
[74] https://developpaper.com/transfer-learning-nlp-visual-diagrams-of-bert
[75] https://medium.com/why-bert-has-3-embedding-layers-and-their-implementation-details-9c261108e28a
[76] https://shapenet.org/
[77] https://stackoverflow.com/questions/58329059/keras-autoencoder-validation-loss-training-loss-but-performing-well-on-tes
[78] Text2Shape: Generating Shapes from Natural Language by Learning Joint Embeddings, https://arxiv.org/abs/1803.08495
[79] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, Honglak Lee (2016) : Generative Adversarial Text to Image Synthesis https://arxiv.org/abs/1605.05396