

Video Summarization Algorithm Using Attention Driven GAN Architecture

MINOR PROJECT REPORT

Submitted in partial fulfilment of the requirements for

the award of the degree

of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE & ENGINEERING

By

**Bharat Goel
01196302720**

**Kunal
02396302720**

**Shresth Kumar
03896302720**

Guided by

**Dr. Neeti Sangwan
Assistant Professor**



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
MAHARAJA SURAJMAL INSTITUTE OF TECHNOLOGY
(AFFILIATED TO GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY, DELHI)
DELHI – 110058**

December 2023

CANDIDATE’S DECLARATION

It is hereby certified that the work which is being presented in the B. Tech Minor Project Report entitled **Video summarization Algorithm Using Attention Driven GAN Architecture** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** and submitted in the **Department of Computer Science & Engineering** of **MAHARAJA SURAJMAL INSTITUTE OF TECHNOLOGY, New Delhi (Affiliated to Guru Gobind Singh Indraprastha University, Delhi)** is an authentic record of our own work carried out during a period from **August 2023 to December 2023** under the guidance of **Dr. Neeti Sangwan, Assistant Professor.**

The matter presented in the B. Tech. Minor Project Report has not been submitted by me for the award of any other degree of this or any other Institute.

Bharat Goel
01196302720

Kunal
02396302720

Shresth Kumar
03896302720

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge. They are permitted to appear in the External Minor Project Examination.

Dr. Neeti Sangwan
Associate Professor, CSE Department
Mentor

Ms. Gunjan Beniwal
Assistant Professor, CSE Department
Project Coordinator

Dr. Nishtha Jatana
HoD, CSE II Shift

Acknowledgment

We express our deep gratitude to **Dr. Neeti Sangwan**, Assistant Professor, Department of Computer Science & Engineering for her valuable guidance and suggestion throughout our project work. We are thankful to **Ms. Gunjan Beniwal**, Project Coordinator, for their valuable guidance.

We would like to extend my sincere thanks to **Head of the Department, Dr. Nishtha Jatana** for her time-to-time suggestions to complete our project work. We are also thankful to **Prof. Archana Balyan, Director MSIT** for providing us the facilities to carry out our project work.

Bharat Goel
01196302720

Kunal
02396302720

Shresth Kumar
03896302720

Table of Content

CANDIDATE DECLARATION	ii
CERTIFICATE	iii
ACKNOWLEDGEMENT	iv
TABLE OF CONTENTS	v
LIST OF FIGURES	ix
LIST OF TABLES	x
ABSTRACT	xi
Chapter 1: Introduction	1
1.1 Overview	1
1.1.1 Video Summarization	1
1.1.2 GAN	4
1.2 Motivation	6
1.3 Objectives	6
1.4 Significance	6
1.5 Summary	7
Chapter 2: Literature Survey	8
Chapter 3: Methodology For Development	11
3.1 RNN	11
3.1.1 Architecture Of Recurrent Neural Network	12
3.1.2 Backpropagation Through Time (BPTT)	13
3.1.3 Training through RNN	15
3.1.4 Advantages of Recurrent Neural Network	16
3.1.5 Disadvantages of Recurrent Neural Network	16
3.1.6 Applications of Recurrent Neural Network	16
3.1.7 Types Of RNN	16
3.1.8 Variation Of Recurrent Neural Network (RNN)	19
3.2 GAN	19

3.2.1	Architecture of GAN	20
3.2.2	GAN working	22
3.2.3	Different Types of GAN Models	23
3.2.4	Application of GANs	24
3.2.5	Advantages of GANs	24
3.2.6	Disadvantages of GANs	25
3.3	TVSum Dataset	25
3.3.1	Purpose	25
3.3.2	Concept	26
3.3.3	Video Sources 8	26
3.3.4	Annotation	26
3.3.5	Video Length	26
3.3.6	Evaluation Metrics	26
3.3.7	Splitting	27
3.3.8	Challenges	27
3.4	JSON	27
3.4.1	Syntax	28
3.4.2	Example JSON Document	29
3.4.3	Use Cases	29
3.4.4	Parsing and Generating JSON	29
3.4.5	JSON Schema	29
3.4.6	Limitations	29
3.4.7	Security Considerations	29
3.4.8	Advantages	30
3.5	Python	30
3.5.1	General Characteristics	30
3.5.2	Syntax	30
3.5.3	Data Structures	30
3.5.4	Control Flow	31
3.5.5	Functions	31
3.5.6	Modules and Libraries	31
3.5.7	Object-Oriented Programming (OOP)	31

3.5.8	File Handling	31
3.5.9	Community and Documentation	31
3.5.10	Applications	31
3.6	PyTorch	32
3.6.1	Tensor Basics	32
3.6.2	Dynamic Computational Graph	33
3.6.3	Automatic Differentiation	33
3.6.4	Neural Networks	33
3.6.5	Data Handling with torch.utils.data	33
3.6.6	Device Management	33
3.6.7	Extensive Community and Ecosystem	33
3.6.8	Research and Production	33
3.6.9	Community and Documentation	33
3.6.10	Ease of Use and Flexibility	34
3.7	CUDA	34
3.7.1	GPU Computing	35
3.7.2	CUDA Programming Model	35
3.7.3	Device Memory	35
3.7.4	Execution Flow	35
3.7.5	CUDA Libraries	35
3.7.6	CUDA Toolkit	36
3.7.7	GPU Architecture	36
3.7.8	Applications of CUDA	36
3.7.9	Limitations and Considerations	36
3.7.10	CUDA Community and Resources	36
3.8	Anaconda	37
3.8.1	Overview	38
3.8.2	Key Components	38
3.8.3	Environments	38
3.8.4	Package Management	39
3.8.5	Data Science Libraries	39
3.8.6	Machine Learning Libraries	39

3.8.7	Interoperability	40
3.8.8	Community and Support	40
3.8.9	Education and Training	40
3.8.10	Deployment and Scaling	38
Chapter 4: Results		41
Chapter 5: Conclusion and future Scope		45
5.1	Conclusion	45
5.2	Scope of Future	45
References		48

List of Figures

3.1	Recurrent Neural Network	11
3.2	What is Recurrent Neural Network	12
3.3	Backpropagation Through Time (BPTT) In RNN	14
3.4	One to One	17
3.5	One to Many	17
3.6	Many to One	18
3.7	Many to Many	18
3.8	GAN Architecture	22
3.9	TVSum	26
3.10	JSON	28
3.11	Pytorch	32
3.12	Pytorch Architecture	32
3.13	CUDA	34
3.14	Anaconda Navigator	38
4.1	Training Step using Splits	41
4.2	Training Step using Split 0	41
4.3	Loss curves of the discriminator (LORIG, LSUM) and generator (LGEN) of SUM-GAN-AAE model.	42
4.4	The proposed SUM-GAN-AAE architecture	43

List of Tables

4.1	Performance (FScore (%)) of SUM-GANAAE for different values of the regularization factor.	42
4.2	Comparison (F-Score (%)) with different unsupervised video summarization approaches, on SumMe and TVSum.	42
4.3	Comparison (F-Score (%)) of our unsupervised method with supervised video summarization approaches on SumMe and TVSum.	43
4.4	Comparison (FScore (%)) of the best performing SUM-GAN model (based on [16]) with the proposed model for different values of the regularization term σ .	43

ABSTRACT

This report outlines the development and evaluation of a Video Summarization Algorithm utilizing an Attention-Driven Generative Adversarial Network (GAN) architecture. With the escalating volume of digital video content, there is a pressing need for efficient methods to distill crucial information. Our algorithm employs a two-step process, integrating an attention mechanism to dynamically highlight pertinent frames and a GAN architecture to generate coherent and representative video summaries. The attention mechanism enables adaptive frame selection based on importance, capturing essential temporal and spatial information. Extensive experiments on diverse video datasets demonstrate the algorithm's superior performance, assessed through quantitative metrics like precision, recall, and F1 score, as well as qualitative analysis, including visual comparisons and user studies. The results affirm the algorithm's effectiveness in generating concise and informative video summaries across various genres. This approach contributes to advancements in content indexing, video browsing, and real-time video analysis. The flexible and adaptable nature of our algorithm positions it as a significant stride in addressing the challenges posed by the exponential growth of video content, marking a noteworthy contribution to the field of video summarization.

Chapter 1 – Introduction

1.1 Overview

The exponential growth of digital video content across various platforms has led to a pressing need for sophisticated video summarization techniques. Video summarization involves condensing lengthy videos into concise representations while retaining the essential information. Traditional methods often struggle with the dynamic and diverse nature of video content. In response to these challenges, our project introduces a groundbreaking approach—Video Summarization via Attention-Driven Adversarial Learning.

1.1.1 Video Summarization

Video summarization is the process of creating a concise and meaningful summary of a longer video while retaining its key content and information. The goal is to condense the video into a shorter version that captures the essential aspects, making it easier for users to quickly grasp the main points without watching the entire video. Video summarization techniques can be applied to various types of videos, including surveillance footage, educational videos, news broadcasts, and more.

There are two primary types of video summarization:

- **Keyframe Summarization**

In keyframe summarization, representative frames are selected from the video to create a summary. These frames are chosen based on certain criteria, such as visual significance, content diversity, or changes in scene or activity.

Keyframes serve as snapshots that convey the essential information of a video, allowing viewers to get a quick overview without watching the entire sequence.

- **Video Skim Summarization**

Video skim summarization involves the extraction of short video clips or segments that collectively represent the important content of the original video.

These video skims are typically longer than individual keyframes and may capture dynamic changes or events within the video.

Video summarization can be performed using various techniques and algorithms, including:

- **Content-Based Methods**

These methods analyze the visual and audio content of the video to identify key features, objects, or events. Techniques such as shot boundary detection, object recognition, and activity recognition may be employed.

- **Text-Based Methods**

Text-based methods involve analyzing textual information associated with the video, such as transcripts, subtitles, or metadata. Natural language processing techniques may be used to extract important keywords and phrases.

- **Hybrid Methods**

Hybrid approaches combine both content-based and text-based methods to leverage both visual and textual information for more accurate summarization.

- **Deep Learning Approaches**

With the advancements in deep learning, convolutional neural networks (CNNs) and recurrent neural networks (RNNs) have been applied to video summarization tasks. These models can automatically learn relevant features and patterns from the data.

Video summarization has practical applications in various domains, including:

- **Surveillance:** Summarizing hours of surveillance footage to highlight critical events.
- **Education:** Creating condensed versions of educational videos for quicker comprehension.
- **News:** Generating brief summaries of news broadcasts for quick updates.
- **Social Media:** Providing users with summarized versions of longer videos shared on platforms like YouTube or Facebook.

Despite the progress made in video summarization, it remains a challenging research area due to the complexity of video content and the subjective nature of what constitutes important information. Ongoing research aims to improve the accuracy and efficiency of video summarization techniques, making them more accessible and applicable to a wide range of video content.

Unsupervised Learning

Unsupervised learning is a type of machine learning paradigm where the algorithm is tasked with extracting patterns or relationships from input data without explicit supervision or labeled outputs. Unlike supervised learning, where the algorithm is provided with labeled

training data to learn from, unsupervised learning involves working with unlabeled data and aims to uncover the inherent structure within that data.

There are two main types of unsupervised learning:

Clustering

Clustering algorithms group similar data points together based on some similarity metric. The goal is to discover natural groupings or clusters within the data.

Common clustering algorithms include k-means clustering, hierarchical clustering, and DBSCAN (Density-Based Spatial Clustering of Applications with Noise).

Dimensionality Reduction

Dimensionality reduction techniques aim to reduce the number of features or variables in the data while preserving its essential characteristics. This is particularly useful for handling high-dimensional data or reducing computational complexity.

Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE) are examples of dimensionality reduction techniques.

Key Concepts in Unsupervised Learning

- **No Labeled Output:** In unsupervised learning, the algorithm does not have access to labeled outputs or target values. It must find patterns, relationships, or structures within the data without explicit guidance.
- **Exploratory Analysis:** Unsupervised learning is often used for exploratory data analysis, where the goal is to understand the underlying patterns or groupings within the data.
- **Anomaly Detection:** Unsupervised learning can be applied to detect anomalies or outliers in the data—instances that deviate significantly from the norm.
- **Representation Learning:** Some unsupervised learning methods focus on learning useful representations of the input data, which can be beneficial for subsequent tasks such as classification or regression.

Applications of Unsupervised Learning

- **Customer Segmentation:** Clustering techniques can be employed to group customers with similar purchasing behavior for targeted marketing strategies.

- **Image and Speech Recognition:** Unsupervised learning can aid in extracting meaningful features from raw image or speech data, contributing to improved recognition systems.
- **Anomaly Detection in Cybersecurity:** Unsupervised learning is often used to identify unusual patterns in network traffic, helping to detect potential security threats.
- **Topic Modeling in Natural Language Processing (NLP):** Techniques like Latent Dirichlet Allocation (LDA) are used for unsupervised learning in NLP to discover topics within a collection of documents.
- **Gene Expression Analysis:** Unsupervised learning methods can reveal patterns in gene expression data, aiding in the identification of functional relationships among genes.

Unsupervised learning plays a crucial role in scenarios where labeled data is scarce or expensive to obtain. It allows for the exploration and understanding of data structures, leading to valuable insights and improved decision-making in various fields.

1.1.2 GAN

GAN, which stands for Generative Adversarial Network, is a class of artificial intelligence algorithms used in unsupervised machine learning. GANs were introduced by Ian Goodfellow and his colleagues in 2014 and have since become a powerful tool for generating realistic synthetic data.

Key Concepts of GAN

Generative Model

GANs are generative models, meaning they are designed to generate new data instances that resemble a given dataset. They can be used to create realistic images, videos, text, and more.

Adversarial Process

The core idea behind GANs is an adversarial process involving two neural networks – a generator and a discriminator – which are trained simultaneously through a competitive process.

The generator creates synthetic data, and the discriminator evaluates whether the data is real or fake.

- **Generator:** The generator's role is to create realistic data that is indistinguishable from the real data in the training set. It takes random noise as input and transforms it into data samples.
- **Discriminator:** The discriminator's role is to distinguish between real and fake data. It is trained on both real and synthetic data and improves its ability to differentiate over time.

Training Process

During training, the generator and discriminator are in a constant loop of improvement. The generator aims to create more realistic data to fool the discriminator, while the discriminator refines its ability to distinguish between real and fake data.

Equilibrium and Convergence

Ideally, GANs reach a point where the generator produces data that is so realistic that the discriminator can no longer differentiate between real and synthetic samples. This equilibrium is known as convergence.

Loss Function: GANs use a specific loss function called the adversarial loss or minimax loss. The generator seeks to minimize this loss, while the discriminator aims to maximize it.

Applications of GAN

- **Image Generation:** GANs can generate realistic images, leading to applications in art creation, face generation, and the synthesis of realistic scenes.
- **Style Transfer:** GANs are used for transferring styles between images, allowing for the transformation of photographs into artwork with a specific artistic style.
- **Data Augmentation:** GANs can be employed to augment training datasets, creating additional synthetic data to enhance the performance of machine learning models.
- **Super-Resolution:** GANs can improve the resolution of images, allowing for the generation of high-quality images from lower-resolution inputs.
- **Image-to-Image Translation:** GANs can be applied to tasks such as converting satellite images to maps, turning black-and-white photos into colour, and more.
- **Drug Discovery:** GANs are used to generate molecular structures, aiding in drug discovery and the development of new pharmaceuticals.

GANs have demonstrated remarkable success in generating realistic and diverse data, but their training can be challenging and requires careful tuning. Researchers continue to explore

variations and improvements to GAN architectures to address limitations and expand their capabilities.

1.2 Motivation

The motivation behind this project is rooted in the recognition that existing video summarization methods face limitations in adaptability and contextual understanding. The emergence of attention mechanisms, inspired by human cognitive processes, presents an opportunity to enhance models' ability to focus on significant video segments. Integrating attention mechanisms with adversarial learning introduces a dynamic and discriminative element to the summarization process, aiming to produce more accurate and contextually rich video summaries.

1.3 Objectives

The primary objectives of our project are multi-faceted:

- **Attention-Driven Feature Extraction:** Implement attention mechanisms to dynamically emphasize salient regions or frames within the video, enabling the model to prioritize crucial content.
- **Adversarial Learning for Discriminative Summarization:** Introduce adversarial learning to augment the discriminative power of the summarization model. The adversarial component challenges the model to generate summaries that are indistinguishable from human-created summaries, promoting higher quality outputs.
- **Contextual Understanding:** Develop a model that can capture the contextual relationships between frames, ensuring that the generated summary maintains coherence and relevance throughout the entire video sequence.

1.4 Significance

This project's significance lies in its potential to advance the field of video summarization by harnessing attention-driven mechanisms and adversarial learning. The resulting model is expected to offer heightened summarization accuracy, capturing nuanced details and relationships within videos that may elude traditional methods. The infusion of adversarial

learning introduces a discriminative element, potentially leading to more contextually relevant and visually coherent video summaries.

1.5 Summary

Our study is a pioneering attempt in pushing the frontiers of video summarization. We hope to construct a model that not only addresses existing issues but also sets a new benchmark for accuracy, contextual comprehension, and adaptability in video summarizing by combining attention-driven processes and adversarial learning. The findings of this study have the potential to have a substantial impact on domains such as video analysis, content retrieval, and information comprehension across a wide range of applications.

Chapter 2 – Literature Survey

P. Saini, K. Kumar, S. Kashid, A. Saini, and A. Negi, et.al(2023)[1] proposed a paper “Video summarization using Deep Learning Techniques: A detailed analysis and investigation,”. In this paper, Detailed analysis and investigation of numerous deep learning approach accomplished to determine root of problems connected with different deep learning methods in identifying and summarizing the essential activities in such videos. Various deep learning tools for experimental analysis have also been discussed in the paper.

Xu Wang , Yujie Li *, Haoyu Wang, Longzhao Huang and Shuxue Ding, et.al. (2022) [2] proposed a paper “A Video Summarization Model Based on Deep Reinforcement Learning with Long-Term Dependency”. In this paper, they proposed AuDSN, which is a deep reinforcement network model with unsupervised auxiliary summarization loss. They introduced unsupervised auxiliary summarization loss in the decoder and explored a novel reward function with a dispersion reward.

Hafiz Burhan Ul Haq, Muhammad Asif , Maaz Bin Ahmad, Rehan Ashraf ,and Toqeer Mahmood , et.al. (2022)[3] proposed a paper “An Effective Video Summarization Framework Based on the Object of Interest Using Deep Learning”. This paper presents an effective VS framework that summarizes the video based on the OoI . The proposed framework is very effective, optimal, and performed much faster than other state-of-the-art methods for summarizing the video .The OoI-based solution makes it more reliable and flexible to generate the relevant video summary.

Negi, K. Kumar, and P. Saini, et.al(2023)[4] proposed a paper “Object of interest and unsupervised learning-based framework for an effective video summarization using Deep Learning,” . The paper introduces a noteworthy framework that combines the identification of the object of interest with unsupervised deep learning techniques to create effective video summaries. This research contributes to the field of video summarization by addressing the challenge of condensing video content into meaningful and informative abstractions, which can have a wide range of applications in information retrieval, content management, and efficient content consumption.

E. Apostolidis, E. Adamantidou, A. I. Metsai, V. Mezaris, and I. Patras, et.al(2019)[5] proposed a paper “Unsupervised video summarization via attention-driven adversarial learning,”. It leverages attention mechanisms and adversarial learning to automatically identify and select important frames or segments in videos to create informative summaries. This research contributes to the development of advanced techniques for video content understanding and has implications for various applications, including video indexing, content retrieval, and efficient video browsing.

M. Basavarajaiah and P. Sharma , et.al(2019)[6] proposed a paper “Survey of compressed domain video summarization techniques,”. It offers insights into the advantages, categorization of techniques, applications, and challenges associated with summarizing videos directly from compressed data. This research contributes to the understanding of efficient video summarization methods, particularly for applications where real-time processing is essential.

M. Basavarajaiah and P. Sharma, et.al(2021)[7] proposed a paper “GVSUM: Generic video summarization using Deep Visual Features,”. This paper builds upon the ideas of [6] and expands the horizons.

J. Fajtl, H. S. Sokeh, V. Argyriou, D. Monekosso, and P. Remagnino, et.al(2019)[8] proposed a paper “Summarizing videos with attention,” . The paper introduces an innovative approach to video summarization by incorporating attention mechanisms, with a focus on temporal attention. This research is valuable for its contributions to the development of more effective and context-aware video summarization techniques, with applications across various domains that involve large volumes of video content.

Evlampios Apostolidis , Eleni Adamantidou , Alexandros I. Metsai , Vasileios Mezaris. et.al(2021)[9] proposed a paper “Video summarization using Deep Neural Networks: A survey” . The paper concludes by summarizing the key takeaways from the survey and emphasizing the growing significance of DNNs in revolutionizing video summarization techniques. It underscores the potential of DNNs to automate and enhance video summarization tasks and encourages further research and innovation in this domain.

Samed Arslan, Senem Tanberk et.al(2023)[10] proposed a paper “Key Frame Extraction with Attention Based Deep Neural Networks” . The primary goal of the research is to develop

an efficient and effective method for automatically identifying key frames within video sequences. Key frames are frames that capture the most important or representative moments in a video and are valuable for various applications, including video summarization, content indexing, and retrieval.

H. Huang and R. K. Wong et.al(2021)[11] proposed a paper “Attention-based SEQ2SEQ regularisation for relation extraction” .This introduces a novel approach to enhance the accuracy and performance of relation extraction models in the field of natural language processing. The authors achieve this by incorporating attention mechanisms and SEQ2SEQ regularization techniques into their model.

Chapter 3 – Methodology For Development

3.1 RNN

Recurrent Neural Network (RNN) is a type of Neural Network where the output from the previous step is fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is its Hidden state, which remembers some information about a sequence. The state is also referred to as Memory State since it remembers the previous input to the network. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks. Figure 3.1 shows a basic RNN visual representation.

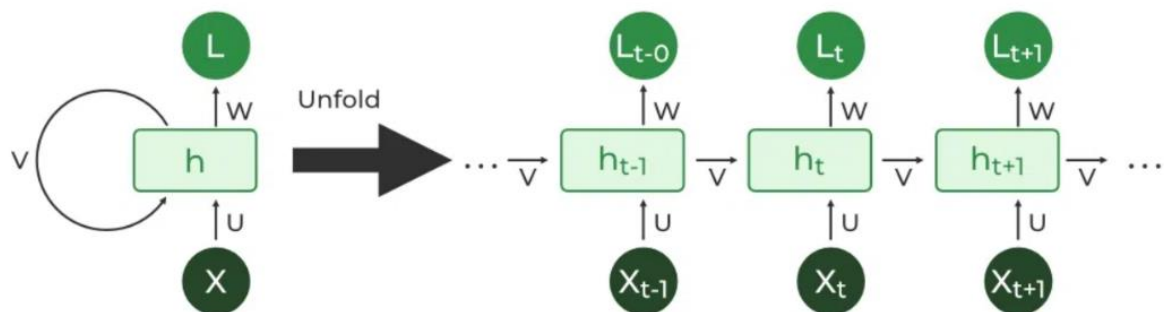


Figure 3.1: Recurrent neural network

3.1.1 Architecture Of Recurrent Neural Network

RNNs have the same input and output architecture as any other deep neural architecture. However, differences arise in the way information flows from input to output. Unlike Deep neural networks where we have different weight matrices for each Dense network in RNN, the weight across the network remains the same. It calculates state hidden state H_i for every

$$\mathbf{h} = \sigma(\mathbf{UX} + \mathbf{W}_{h-1} + \mathbf{B})$$

$$\mathbf{Y} = \mathbf{O}(\mathbf{Vh} + \mathbf{C}) \text{ Hence}$$

$$\mathbf{Y} = \mathbf{f}(\mathbf{X}, \mathbf{h}, \mathbf{W}, \mathbf{U}, \mathbf{V}, \mathbf{B}, \mathbf{C})$$

*Here S is the State matrix which has element s_i as the state of the network at timestep i
The parameters in the network are W, U, V, c, b which are shared across timestep*

input X_i . By using the following formulas:

RECURRENT NEURAL NETWORKS

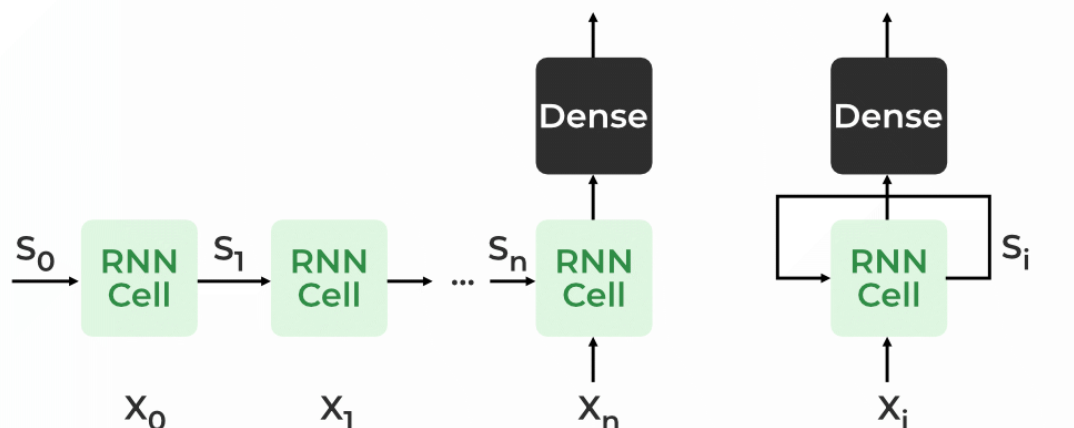


Figure 3.2: What is Recurrent Neural Network

Figure 3.2 gives a basic RNN representation. The Recurrent Neural Network consists of multiple fixed activation function units, one for each time step. Each unit has an internal state which is called the hidden state of the unit. This hidden state signifies the past knowledge that the network currently holds at a given time step. This hidden state is updated at every time step to signify the change in the knowledge of the network about the past. The hidden state is updated using the following recurrence relation:-

The formula for calculating the current state:

$$h_t = f(h_{t-1}, x_t)$$

where:

h_t -> current state

h_{t-1} -> previous state

x_t -> input state

Formula for applying Activation function(tanh):

$$h_t = \tanh (W_{hh}h_{t-1} + W_{xh}x_t)$$

where:

W_{hh} -> weight at recurrent neuron

W_{xh} -> weight at input neuron

The formula for calculating output:

$$y_t = W_{hy}h_t$$

Y_t -> output

W_{hy} -> weight at output layer

These parameters are updated using Backpropagation. However, since RNN works on sequential data here we use an updated backpropagation which is known as Backpropagation through time.

3.1.2 Backpropagation Through Time (BPTT)

In RNN the neural network is in an ordered fashion and since in the ordered network each variable is computed one at a time in a specified order like first h_1 then h_2 then h_3 so on. Hence ,we will apply backpropagation throughout all these hidden time states sequentially which is shown in Figure 3.3. Backpropagation Through Time (BPTT) In RNN

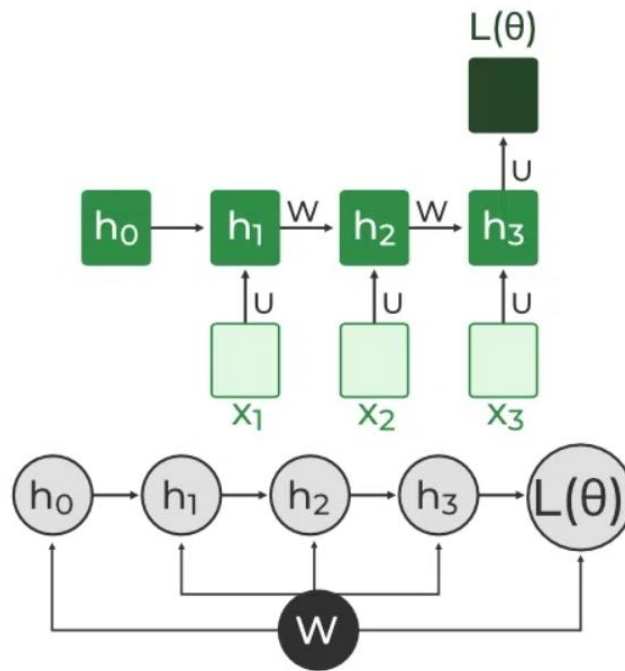


Figure 3.3: Backpropagation Through Time (BPTT) In RNN

$L(\theta)$ (loss function) depends on h_3

h_3 in turn depends on h_2 and W

h_2 in turn depends on h_1 and W

h_1 in turn depends on h_0 and W

where h_0 is a constant starting state.

$$\frac{\partial L(\theta)}{\partial W} = \sum_{t=1}^T \frac{\partial L(\theta)}{\partial W}$$

For simplicity of this equation, we will apply backpropagation on only one row

$$\frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial h_3} \frac{\partial h_3}{\partial W}$$

We already know how to compute this one as it is the same as any simple deep neural network backpropagation .However, we will see how to apply backpropagation to this term

As we know $h_3 = \sigma(W h_2 + b)$

And In such an ordered network, we can't compute $\frac{\partial h_3}{\partial W}$ by simply treating h_3 as a constant

because as it also depends on W . the total derivative $\frac{\partial h_3}{\partial W}$ has two parts

Explicit: $\frac{\partial h_3}{\partial W}$ treating all other inputs as constant

Implicit: Summing over all indirect paths from h_3 to W

Let us see how to do this

$$\begin{aligned}
 \frac{\partial h_3}{\partial W} &= \frac{\partial h_3^+}{\partial W} + \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial W} \\
 &= \frac{\partial h_3^+}{\partial W} + \frac{\partial h_3}{\partial h_2} \left[\frac{\partial h_2^+}{\partial W} + \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial W} \right] \\
 &= \frac{\partial h_3^+}{\partial W} + \frac{\partial h_3}{\partial h_2} \frac{\partial h_2^+}{\partial W} + \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1} \left[\frac{\partial h_1^+}{\partial W} \right]
 \end{aligned}$$

For simplicity, we will short-circuit some of the paths

$$\frac{\partial h_3}{\partial W} = \frac{\partial h_3^+}{\partial W} + \frac{\partial h_3}{\partial h_2} \frac{\partial h_2^+}{\partial W} + \frac{\partial h_3}{\partial h_1} \frac{\partial h_1^+}{\partial W}$$

Finally, we have

$$\frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial h_3} \cdot \frac{\partial h_3}{\partial W}$$

Where

$$\frac{\partial h_3}{\partial W} = \sum_{k=1}^3 \frac{\partial h_3}{\partial h_k} \cdot \frac{\partial h_k}{\partial W}$$

Hence,

$$\frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial h_3} \sum_{k=1}^3 \frac{\partial h_3}{\partial h_k} \cdot \frac{\partial h_k}{\partial W}$$

This algorithm is called backpropagation through time (BPTT) as we backpropagate over all previous time steps

3.1.3 Training through RNN

- 1 A single-time step of the input is provided to the network.
- 2 Then calculate its current state using a set of current input and the previous state.
- 3 The current h_t becomes h_{t-1} for the next time step.
- 4 One can go as many time steps according to the problem and join the information from all the previous states.
- 5 Once all the time steps are completed the final current state is used to calculate the output.
- 6 The output is then compared to the actual output i.e the target output and the error is generated.

- 7 The error is then back-propagated to the network to update the weights and hence the network (RNN) is trained using Backpropagation through time.

3.1.4 Advantages of Recurrent Neural Network

- An RNN remembers each and every piece of information through time. It is useful in time series prediction only because of the feature to remember previous inputs as well. This is called Long Short Term Memory.
- Recurrent neural networks are even used with convolutional layers to extend the effective pixel neighbourhood .

3.1.5 Disadvantages of Recurrent Neural Network

- Gradient vanishing and exploding problems.
- Training an RNN is a very difficult task.
- It cannot process very long sequences if using tanh or relu as an activation function.

3.1.6 Applications of Recurrent Neural Network

- ❖ Language Modelling and Generating Text
- ❖ Speech Recognition
- ❖ Machine Translation
- ❖ Image Recognition, Face detection
- ❖ Time series Forecasting

3.1.7 Types Of RNN

There are four types of RNNs based on the number of inputs and outputs in the network.

One to One

This type of RNN behaves the same as any simple Neural network it is also known as Vanilla Neural Network. In this Neural network, there is only one input and one output. Figure 3.4 represents a One to One RNN.

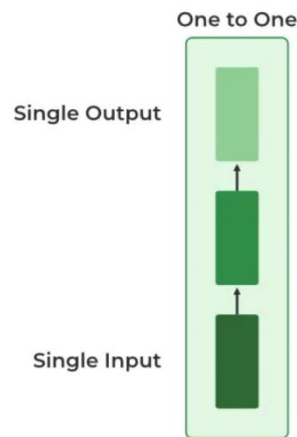


Figure 3.4: One to One RNN

One To Many

In this type of RNN, there is one input and many outputs associated with it. One of the most used examples of this network is Image captioning where given an image we predict a sentence having Multiple words. Figure 3.5 represents a One to Many RNN.

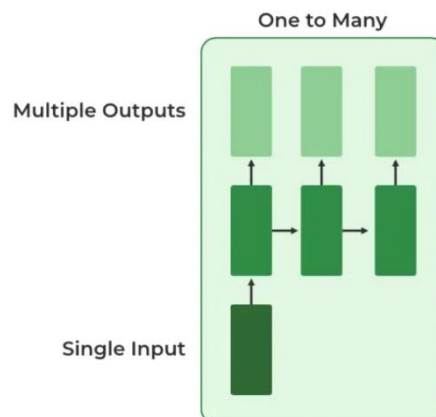


Figure 3.5: One to Many RNN

Many to One

In this type of network, Many inputs are fed to the network at several states of the network generating only one output. This type of network is used in the problems like sentimental analysis. Where we give multiple words as input and predict only the sentiment of the sentence as output. Figure 3.6 represents a Many to One RNN.

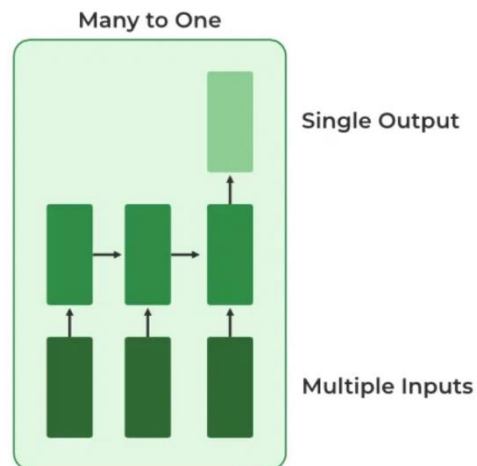


Figure 3.6: Many to One RNN

Many to Many

In this type of neural network, there are multiple inputs and multiple outputs corresponding to a problem. One Example of this Problem will be language translation. In language translation, we provide multiple words from one language as input and predict multiple words from the second language as output. Figure 3.7 represents a Many to Many RNN.

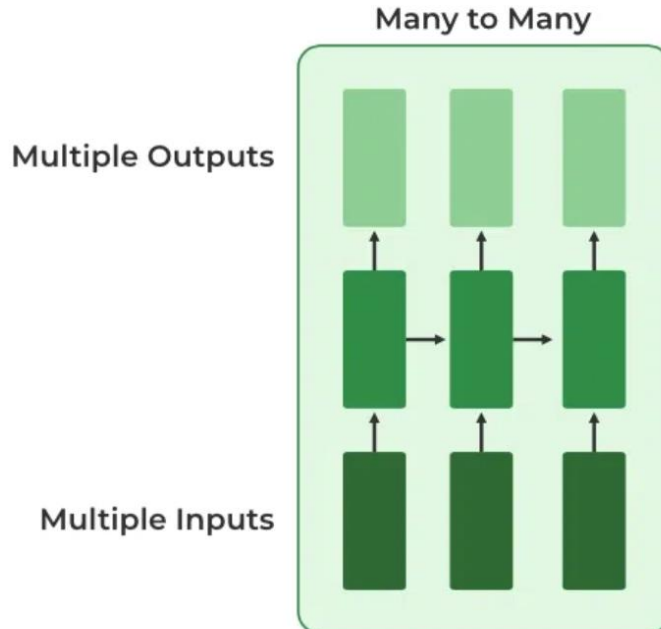


Figure 3.7: Many to Many RNN

3.1.8 Variation Of Recurrent Neural Network (RNN)

To overcome the problems like vanishing gradient and exploding gradient descent several new advanced versions of RNNs are formed some of these are as ;

- Bidirectional Neural Network (BiNN)
- Long Short-Term Memory (LSTM)

3.2 GAN

Generative Adversarial Networks, or GANs, represent a cutting-edge approach to generative modeling within deep learning, often leveraging architectures like convolutional neural networks. The goal of generative modeling is to autonomously identify patterns in input data, enabling the model to produce new examples that feasibly resemble the original dataset.

GANs tackle this challenge through a unique setup, treating it as a supervised learning problem involving two key components: the generator, which learns to produce novel examples, and the discriminator, tasked with distinguishing between genuine and generated instances. Through adversarial training, these models engage in a competitive interplay until the generator becomes adept at creating realistic samples, fooling the discriminator approximately half the time.

This dynamic field of GANs has rapidly evolved, showcasing remarkable capabilities in generating lifelike content across various domains. Notable applications include image-to-image translation tasks and the creation of photorealistic images indistinguishable from real photos, demonstrating the transformative potential of GANs in the realm of generative modeling.

Generative Adversarial Networks (GANs) are a powerful class of neural networks that are used for unsupervised learning. GANs are made up of two neural networks, a discriminator and a generator. They use adversarial training to produce artificial data that is identical to actual data. The Generator attempts to fool the Discriminator, which is tasked with accurately distinguishing between produced and genuine data, by producing random noise samples. Realistic, high-quality samples are produced as a result of this competitive interaction, which drives both networks toward advancement. GANs are proving to be highly versatile artificial intelligence tools, as evidenced by their extensive use in image synthesis, style transfer, and text-to-image synthesis. They have also revolutionized generative modeling.

Generative Adversarial Networks (GANs) can be broken down into three parts:

- Generative: To learn a generative model, which describes how data is generated in terms of a probabilistic model.
- Adversarial: The word adversarial refers to setting one thing up against another. This means that, in the context of GANs, the generative result is compared with the actual images in the data set. A mechanism known as a discriminator is used to apply a model that attempts to distinguish between real and fake images.
- Networks: Use deep neural networks as artificial intelligence (AI) algorithms for training purposes.

3.2.1 Architecture of GAN

A Generative Adversarial Network (GAN) is composed of two primary parts, which are the Generator and the Discriminator.

Generator Model

A key element responsible for creating fresh, accurate data in a Generative Adversarial Network (GAN) is the generator model. The generator takes random noise as input and converts it into complex data samples, such text or images. It is commonly depicted as a deep neural network. The training data's underlying distribution is captured by layers of learnable parameters in its design through training. The generator adjusts its output to produce samples that closely mimic real data as it is being trained by using backpropagation to fine-tune its parameters. The generator's ability to generate high-quality, varied samples that can fool the discriminator is what makes it successful.

Generator Loss(J_G)

For generated samples, the generator minimizes the log likelihood that the discriminator is right. Due to this loss, the generator is incentivized to generate samples that the discriminator is likely to classify as real ($\log D(G(z_i))$ close to 1).

$$J_G = -\frac{1}{m} \sum_{i=1}^m \log D(G(z_i))$$

Where ,

- J_G measure how well the generator is fooling the discriminator.

- $\log D(G(z_i))$ represents log probability of the discriminator being correct for generated samples.
- The generator aims to minimize this loss, encouraging the production of samples that the discriminator classifies as real ($\log D(G(z_i))$ close to 1).

Discriminator Model

An artificial neural network called a discriminator model is used in Generative Adversarial Networks (GANs) to differentiate between generated and actual input. By evaluating input samples and allocating probability of authenticity, the discriminator functions as a binary classifier. Over time, the discriminator learns to differentiate between genuine data from the dataset and artificial samples created by the generator. This allows it to progressively hone its parameters and increase its level of proficiency. Convolutional layers or pertinent structures for other modalities are usually used in its architecture when dealing with picture data. Maximizing the discriminator's capacity to accurately identify generated samples as fraudulent and real samples as authentic is the aim of the adversarial training procedure. The discriminator grows increasingly discriminating as a result of the generator and discriminator's interaction, which helps the GAN produce extremely realistic-looking synthetic data overall.

Discriminator Loss(J_D)

The discriminator reduces the negative log likelihood of correctly classifying both produced and real samples. This loss incentivizes the discriminator to accurately categorize generated samples as fake ($\log(1-D(G(z_i)))$ close to 1) and real samples ($\log D(x_i)$ close to 1).

$$J_D = -\frac{1}{m} \sum_{i=1}^m \log D(x_i) - \frac{1}{m} \sum_{i=1}^m \log(1 - D(G(z_i)))$$

J_D assesses the discriminator's ability to discern between produced and actual samples.

- The log likelihood that the discriminator will accurately categorize real data is represented by $\log D(x_i)$.
- The log chance that the discriminator would correctly categorize generated samples as fake is represented by $\log_{\{f_0\}}(1-D(G(z_i)))$.
- The discriminator aims to reduce this loss by accurately identifying artificial and real samples.

MinMax Loss

In a Generative Adversarial Network (GAN), the minimax loss formula is provided by:

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

Where,

- G is generator network and D is the discriminator network
- Actual data samples obtained from the true data distribution $p_{data}(x)$ are represented by x.
- Random noise sampled from a previous distribution $p_z(z)$ (usually a normal or uniform distribution) is represented by z.
- $D(x)$ represents the discriminator's likelihood of correctly identifying actual data as real.
- $D(G(z))$ is the likelihood that the discriminator will identify generated data coming from the generator as authentic.

Figure 3.8 represents in GAN architecture.

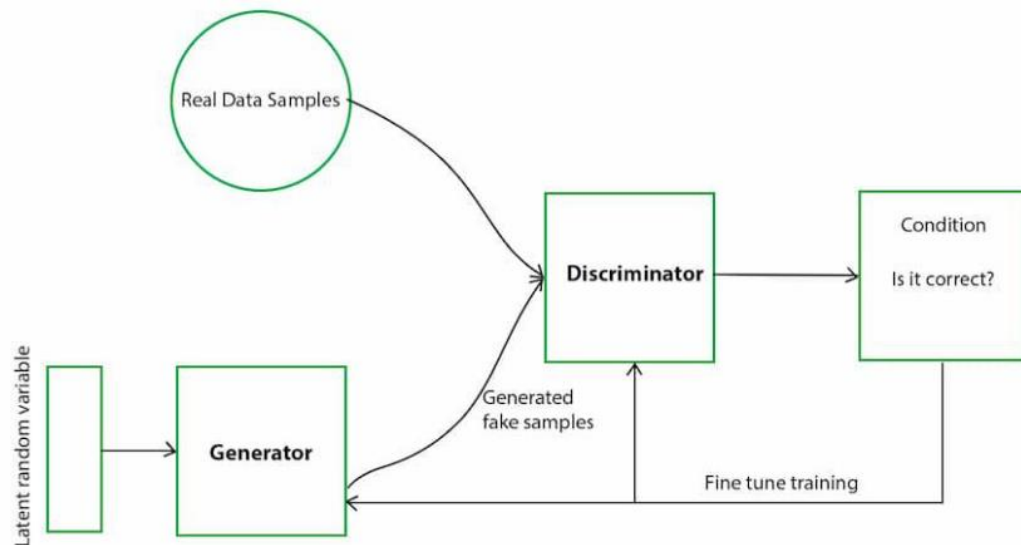


Figure 3.8: GAN Architecture

3.2.2 GAN working

A Generative Adversarial Network (GAN) is a framework made up of two neural networks that have undergone simultaneous adversarial training—a discriminator and a generator. The discriminator separates generated data from real data, while the generator produces synthetic

data that attempts to imitate real data. Training makes the generator more adept at producing realistic samples in an effort to trick the discriminator, which strengthens the generator's discriminating abilities. GANs are an effective tool for producing realistic, high-quality outputs in a variety of fields, including text and image generation, because of this back-and-forth competition, which results in the creation of increasingly convincing and indistinguishable synthetic data.

3.2.3 Different Types of GAN Models

- 1 Vanilla GAN: This is the simplest type of GAN. Here, the Generator and the Discriminator are simple multi-layer perceptrons. In vanilla GAN, the algorithm is really simple, it tries to optimize the mathematical equation using stochastic gradient descent.
- 2 Conditional GAN (CGAN): CGAN can be described as a deep learning method in which some conditional parameters are put into place. In CGAN, an additional parameter 'y' is added to the Generator for generating the corresponding data. Labels are also put into the input to the Discriminator in order for the Discriminator to help distinguish the real data from the fake generated data.
- 3 Deep Convolutional GAN (DCGAN): DCGAN is one of the most popular and also the most successful implementations of GAN. It is composed of ConvNets in place of multi-layer perceptrons. The ConvNets are implemented without max pooling, which is in fact replaced by convolutional stride. Also, the layers are not fully connected.
- 4 Laplacian Pyramid GAN (LAPGAN): The Laplacian pyramid is a linear invertible image representation consisting of a set of band-pass images, spaced an octave apart, plus a low-frequency residual. This approach uses multiple numbers of Generator and Discriminator networks and different levels of the Laplacian Pyramid. This approach is mainly used because it produces very high-quality images. The image is down-sampled at first at each layer of the pyramid and then it is again up-scaled at each layer in a backward pass where the image acquires some noise from the Conditional GAN at these layers until it reaches its original size.
- 5 Super Resolution GAN (SRGAN): SRGAN as the name suggests is a way of designing a GAN in which a deep neural network is used along with an adversarial network in order to produce higher-resolution images. This type of GAN is particularly useful in optimally up-scaling native low-resolution images to enhance their details minimizing errors while doing so.

3.2.4 Application Of Generative Adversarial Networks (GANs)

GANs, or Generative Adversarial Networks, have many uses in many different fields. Here are some of the widely recognized uses of GANs:

- ❖ **Image Synthesis and Generation** : GANs are often used for picture synthesis and generation tasks. They may create fresh, lifelike pictures that mimic training data by learning the distribution that explains the dataset. The development of lifelike avatars, high-resolution photographs, and fresh artwork have all been facilitated by these types of generative networks.
- ❖ **Image-to-Image Translation** : GANs may be used for problems involving image-to-image translation, where the objective is to convert an input picture from one domain to another while maintaining its key features. GANs may be used, for instance, to change pictures from day to night, transform drawings into realistic images, or change the creative style of an image.
- ❖ **Text-to-Image Synthesis** : GANs have been used to create visuals from descriptions in text. GANs may produce pictures that translate to a description given a text input, such as a phrase or a caption. This application might have an impact on how realistic visual material is produced using text-based instructions.
- ❖ **Data Augmentation** : GANs can augment present data and increase the robustness and generalizability of machine-learning models by creating synthetic data samples.
- ❖ **Data Generation for Training** : GANs can enhance the resolution and quality of low-resolution images. By training on pairs of low-resolution and high-resolution images, GANs can generate high-resolution images from low-resolution inputs, enabling improved image quality in various applications such as medical imaging, satellite imaging, and video enhancement.

3.2.5 Advantages of Generative Adversarial Networks (GANs)

The advantages of the GANs are as follows:

- ❖ **Synthetic data generation**: GANs can generate new, synthetic data that resembles some known data distribution, which can be useful for data augmentation, anomaly detection, or creative applications.
- ❖ **High-quality results**: GANs can produce high-quality, photorealistic results in image synthesis, video synthesis, music synthesis, and other tasks.

- ❖ Unsupervised learning: GANs can be trained without labeled data, making them suitable for unsupervised learning tasks, where labeled data is scarce or difficult to obtain.
- ❖ Versatility: GANs can be applied to a wide range of tasks, including image synthesis, text-to-image synthesis, image-to-image translation, anomaly detection, data augmentation, and others.

3.2.6 Disadvantages of Generative Adversarial Networks (GANs)

The disadvantages of the GANs are as follows:

- ❖ Training Instability: GANs can be difficult to train, with the risk of instability, mode collapse, or failure to converge.
- ❖ Computational Cost: GANs can require a lot of computational resources and can be slow to train, especially for high-resolution images or large datasets.
- ❖ Overfitting: GANs can overfit the training data, producing synthetic data that is too similar to the training data and lacking diversity.
- ❖ Bias and Fairness: GANs can reflect the biases and unfairness present in the training data, leading to discriminatory or biased synthetic data.
- ❖ Interpretability and Accountability: GANs can be opaque and difficult to interpret or explain, making it challenging to ensure accountability, transparency, or fairness in their applications.

3.3 TVSum Dataset

3.3.1 Purpose

The TVSum dataset is designed for the evaluation and benchmarking of video summarization algorithms. It provides a standardized set of videos with associated human-generated summaries, making it a valuable resource for researchers and practitioners working on video summarization tasks. Figure 3.10 shows a representation of the TVSum dataset.



Figure 3.9 : TVSum

3.3.2 Contents

The dataset comprises a collection of videos extracted from TV shows and movies, covering a diverse range of topics, scenes, and visual content. Each video in the dataset is associated with one or more human-generated summaries.

3.3.3 Video Sources

The videos are sourced from various television shows and movies, ensuring a broad representation of content types and genres. This diversity helps in evaluating the generalizability of video summarization algorithms across different domains.

3.3.4 Annotation

Each video in the TVSum dataset comes with multiple human-generated summaries, providing a range of perspectives on what constitutes a meaningful summary. These summaries serve as ground truth for evaluating the performance of automatic summarization algorithms.

3.3.5 Video Length

Videos in the TVSum dataset vary in length, simulating real-world scenarios where videos can be of different durations. This challenges algorithms to produce concise and informative summaries regardless of the input video's duration.

3.3.6 Evaluation Metrics

Standard evaluation metrics, such as precision, recall, F1-score, and others, are commonly used with the TVSum dataset to assess the performance of video summarization algorithms. These metrics compare the generated summaries with the human-generated summaries to quantify the algorithm's effectiveness.

3.3.7 Splitting

The dataset is typically divided into training, validation, and test sets. This allows researchers to train their models on a subset of the data, tune hyperparameters using the validation set, and evaluate the model's performance on the test set.

3.3.8 Challenges

TVSum poses challenges related to the variability in video content, summarization requirements, and the need for algorithms to generalize well across different genres. This diversity encourages the development of robust and versatile summarization techniques.

- **Research Impact:** Due to its widespread use in the research community, the TVSum dataset has played a crucial role in advancing the state-of-the-art in video summarization. It facilitates fair comparisons between different algorithms and encourages the development of techniques that can handle real-world video data effectively.
- **Access:** The TVSum dataset is often publicly available for research purposes, allowing researchers to download and use it in their experiments. Access may be subject to specific terms and conditions set by the dataset creators.

Researchers and practitioners working on video summarization often leverage benchmark datasets like TVSum to assess the performance of their algorithms, compare against existing methods, and drive advancements in the field.

3.4 JSON

JSON, which stands for JavaScript Object Notation, is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate. JSON is widely used for data exchange between a server and a web application, as well as for configuration files and storing structured data. It is language-independent and supported by a variety of programming languages. Here's a detailed explanation of JSON given in figure 3.10.

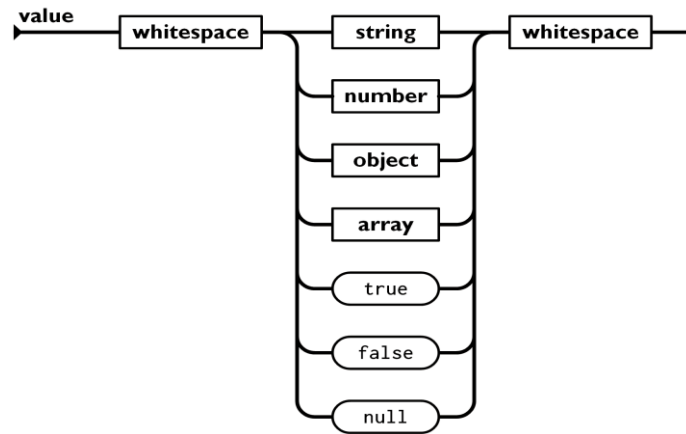


Figure 3.10 : JSON

3.4.1 Syntax

Data Types: JSON supports several data types, including strings, numbers, objects, arrays, booleans, and null.

- Object: An object is an unordered collection of key-value pairs enclosed in curly braces {}. Each key must be a string, and values can be strings, numbers, booleans, objects, arrays, or null.

Json code

```
{ "name": "John Doe", "age": 30, "isStudent": false, "address": { "city": "New York", "zip": "10001" }, "grades": [85, 90, 78] }
```

- Array: An array is an ordered list of values enclosed in square brackets [].

Json code

```
["apple", "banana", "orange"]
```

- String: A string is a sequence of characters enclosed in double quotes " ".

Json code

```
"Hello, World!"
```

- Number: A number can be an integer or a floating-point number.

Json code

```
42
```

- Boolean: Represents true or false.

Json code

```
true
```

- Null: Represents a null value.

Json code

```
null
```

3.4.2 Example JSON Document

Here's an example of a more complex JSON document:

Json code

```
{ "person": { "name": "Alice", "age": 28, "isStudent": false, "address": { "city": "Seattle",  
"zipcode": "98101" }, "hobbies": ["reading", "hiking", "photography"] }, "employees": [ {  
"name": "Bob", "position": "Engineer", "isManager": false }, { "name": "Charlie",  
"position": "Manager", "isManager": true } ] }
```

3.4.3 Use Cases

- **Data Exchange:** JSON is commonly used for exchanging data between a web server and a web client. APIs often return or accept JSON-formatted data.
- **Configuration Files:** JSON is used for configuration files due to its simplicity and human-readable format.
- **Storing and Transmitting Data:** JSON is used to store and transmit structured data between systems.

3.4.4 Parsing and Generating JSON

- **Parsing:** Most programming languages provide built-in functions or libraries to parse JSON into data structures native to that language.
- **Generating:** Similarly, JSON can be generated from data structures in programming languages.

3.4.5 JSON Schema

Validation: JSON Schema provides a way to describe and validate the structure and content of JSON documents.

3.4.6 Limitations

- **No Comments:** JSON does not support comments within the data.
- **No Functions:** JSON is data-only; it does not support executable functions.

3.4.7 Security Considerations

JSON Injection: Developers need to be cautious of JSON injection attacks when dynamically creating JSON.

3.4.8 Advantages

- **Readability:** JSON is easy for humans to read and write.
- **Interoperability:** JSON is language-independent, making it easy to share data between different programming languages.
- **Lightweight:** JSON is a lightweight format, making it efficient for data exchange.

JSON's simplicity, readability, and flexibility make it a popular choice for data interchange in a variety of applications. It has become a de facto standard for web APIs and is widely supported in modern programming languages.

3.5 Python

Python is a high-level, general-purpose programming language known for its simplicity, readability, and versatility. Created by Guido van Rossum and first released in 1991, Python has since become one of the most popular programming languages. Here is a detailed explanation of Python:

3.5.1 General Characteristics

- **High-level Language:** Python is a high-level programming language, meaning it abstracts away many low-level details such as memory management and provides constructs that are easy to understand and use.
- **Interpreted:** Python is an interpreted language, which means that the Python code is executed line by line by an interpreter, making it easy to test and debug code.
- **Readable and Expressive:** Python emphasizes code readability, using a clean and straightforward syntax that allows developers to express concepts in fewer lines of code compared to languages like C++ or Java.

3.5.2 Syntax

- **Indentation:** Python uses indentation (whitespace at the beginning of a line) to indicate blocks of code. This is a distinctive feature of Python and contributes to its readability.
- **Dynamic Typing:** Python is dynamically typed, meaning you don't have to specify the data type of a variable. The interpreter infers it during runtime.

3.5.3 Data Structures

- **Lists:** Ordered, mutable collections.
- **Tuples:** Ordered, immutable collections.

- Dictionaries: Unordered collections of key-value pairs.
- Sets: Unordered collections of unique elements.

3.5.4 Control Flow

- Conditional Statements: if, elif, and else statements for decision-making.
- Loops: for and while loops for iteration.

3.5.5 Functions

- Function Definition: Define reusable blocks of code.
- Lambda Functions: Anonymous functions.

3.5.6 Modules and Libraries

- Modules: Organize Python code into reusable files.
- Libraries: Python has a vast ecosystem of libraries for various purposes, such as NumPy for numerical computing, pandas for data manipulation, and TensorFlow for machine learning.

3.5.7 Object-Oriented Programming (OOP)

Classes and Objects: Python supports object-oriented programming principles.

3.5.8 File Handling

Reading and Writing Files: Python provides functions to read from and write to files.

3.5.9 Community and Documentation

- Python Community: Python has a vibrant and supportive community. The Python Software Foundation (PSF) oversees the development of the language.
- Documentation: Python's official documentation is comprehensive, making it easy for developers to find information about the language and its libraries.

3.5.10 Applications

- Web Development: Frameworks like Django and Flask are widely used for web development.
- Data Science and Machine Learning: Python is a popular choice for data analysis and machine learning tasks with libraries such as NumPy, pandas, scikit-learn, and TensorFlow.
- Scripting and Automation: Python is commonly used for writing scripts and automating repetitive tasks.

- Desktop GUI Applications: Libraries like Tkinter enable the development of graphical user interface (GUI) applications.
- Network Programming: Python provides modules for network programming, making it suitable for building networked applications.

Python's versatility, ease of learning, and extensive libraries contribute to its widespread use across various domains. Whether you're a beginner or an experienced developer, Python provides a powerful and enjoyable programming experience.

3.6 PyTorch

PyTorch is an open-source machine learning library developed by Facebook's AI Research lab (FAIR). It is widely used for deep learning and artificial intelligence applications. PyTorch is known for its dynamic computation graph, which makes it particularly suitable for research and experimentation. Here's a detailed explanation of PyTorch: Figure 3.11 shows the PyTorch logo and figure 3.12 shows the PyTorch Architecture.



Figure 3.11 : PyTorch

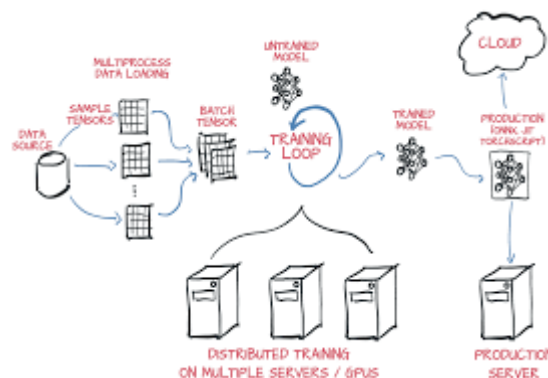


Figure 3.12 : PyTorch Architecture

3.6.1 Tensor Basics

- Tensor: The fundamental data structure in PyTorch is a tensor, which is a multi-dimensional array similar to NumPy arrays.

- Operations: PyTorch provides a wide range of tensor operations, similar to NumPy, allowing mathematical manipulations on tensors.

3.6.2 Dynamic Computational Graph

- Dynamic Graphs: PyTorch uses a dynamic computational graph, allowing for dynamic and on-the-fly graph creation. This is in contrast to static computational graphs used by frameworks like TensorFlow.

3.6.3 Automatic Differentiation

- Gradient Calculation: PyTorch can automatically calculate gradients for tensors using its autograd module.

3.6.4 Neural Networks

- torch.nn Module: PyTorch provides a modular and flexible torch.nn module for building and training neural networks.
- Loss Functions and Optimizers: PyTorch includes various loss functions and optimizers commonly used in training neural networks.

3.6.5 Data Handling with torch.utils.data

- DataLoaders: PyTorch provides utilities for handling datasets and creating data loaders to efficiently load and batch data during training.

3.6.6 Device Management

- GPU Acceleration: PyTorch supports GPU acceleration, allowing for faster training of deep learning models.

3.6.7 Extensive Community and Ecosystem

- Libraries and Extensions: PyTorch has a rich ecosystem, with extensions like torchvision for computer vision tasks, torchtext for natural language processing, and torchaudio for audio processing.

3.6.8 Research and Production

- Research-Friendly: PyTorch's dynamic graph and intuitive design make it well-suited for research and experimentation.
- Production Deployment: PyTorch provides tools like TorchScript for converting models to a serialized format suitable for deployment in production environments.

3.6.9 Community and Documentation

- Community Support: PyTorch has a large and active community of researchers, developers, and practitioners.
- Documentation: PyTorch's official documentation is comprehensive, providing detailed explanations and examples for its various modules and functions.

3.6.10 Ease of Use and Flexibility

- Pythonic Interface: PyTorch follows a Pythonic and intuitive interface, making it easy for users to pick up and start experimenting with deep learning.
- Dynamic Nature: The dynamic nature of PyTorch allows for easier debugging and experimentation during model development.

PyTorch has gained popularity for its flexibility, ease of use, and dynamic nature, particularly in the research community. It is widely adopted in academia and industry, contributing to advancements in the field of deep learning. Whether you're a researcher, a student, or a practitioner, PyTorch provides a powerful and accessible framework for developing and experimenting with deep learning models.

3.7 CUDA

CUDA, which stands for Compute Unified Device Architecture, is a parallel computing platform and application programming interface (API) model created by NVIDIA. It enables developers to harness the computational power of NVIDIA GPUs (Graphics Processing Units) for general-purpose processing tasks, including parallel programming and numerical computations. Figure 3.13 gives visual representation of CUDA. Here's an explanation of CUDA:

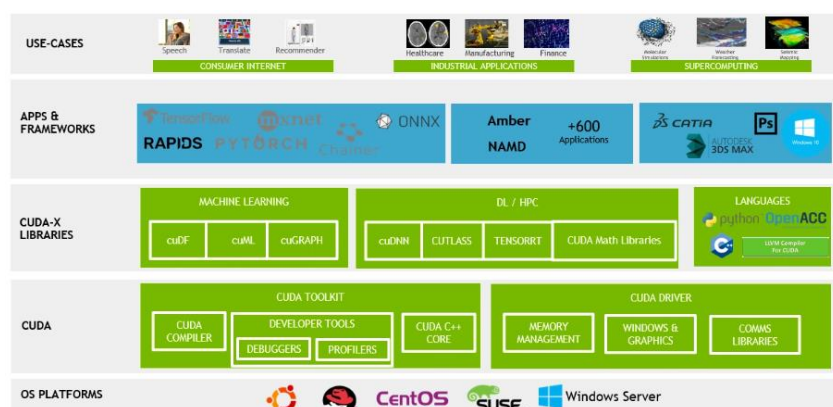


Figure 3.13 : Compute Unified Device Architecture

3.7.1 GPU Computing

- **Parallel Processing:** GPUs are specialized processors designed to handle parallel processing tasks. While traditional CPUs (Central Processing Units) are optimized for sequential processing, GPUs excel at simultaneously executing multiple tasks in parallel.
- **CUDA-enabled GPUs:** To leverage the power of GPUs, CUDA provides a programming model that allows developers to write code that can be executed on NVIDIA GPUs.

3.7.2 CUDA Programming Model

- **Kernels:** In CUDA, the basic unit of execution is called a kernel. A kernel is a function that can be executed in parallel by multiple threads on the GPU.
- **Threads and Blocks:** Kernels are organized into grids, which are made up of blocks. Each block is further divided into threads. Threads within a block can share data and synchronize, while blocks can execute independently.

3.7.3 Device Memory

- **Global Memory:** CUDA provides a hierarchy of memory spaces. Global memory is the main memory space accessible by both the CPU and GPU. It is used for storing data that needs to be accessed by all threads.
- **Shared Memory:** Shared memory is a fast, on-chip memory space that can be used for communication and data sharing among threads within the same block.

3.7.4 Execution Flow

- **Host and Device Code:** CUDA programs typically consist of both host code (executed on the CPU) and device code (executed on the GPU). The host code manages data transfer between the CPU and GPU and launches kernels on the GPU.
- **Data Transfer:** Data needs to be explicitly transferred between the host and device memory. CUDA provides functions like `cudaMemcpy` for managing data transfer.

3.7.5 CUDA Libraries

- cuBLAS and cuFFT: NVIDIA provides libraries like cuBLAS (Basic Linear Algebra Subprograms) and cuFFT (Fast Fourier Transform) that are optimized for GPU acceleration.
- cuDNN: CUDA Deep Neural Network library (cuDNN) is a GPU-accelerated library for deep neural networks, providing optimized primitives for tasks like convolution and pooling.

3.7.6 CUDA Toolkit

- Compilation and Debugging: The CUDA Toolkit includes a compiler (nvcc) for compiling CUDA code and a debugger (cuda-gdb) for debugging GPU-accelerated applications.

3.7.7 GPU Architecture

- Streaming Multiprocessors (SMs): Modern NVIDIA GPUs are composed of multiple streaming multiprocessors, each capable of executing multiple threads concurrently.
- Warp and Thread Execution: Threads are grouped into warps, and warps are scheduled for execution on SMs. The GPU hardware manages the execution of these threads in parallel.

3.7.8 Applications of CUDA

- Parallel Computing: CUDA is widely used for parallel computing tasks, including scientific simulations, numerical simulations, and data processing.
- Deep Learning: Many deep learning frameworks, such as TensorFlow and PyTorch, use CUDA for GPU acceleration to train and deploy neural networks efficiently.
- Scientific Computing: CUDA is commonly used in scientific research and simulations due to its ability to accelerate complex computations.
- Image and Signal Processing: Applications involving image and signal processing benefit from CUDA's parallel processing capabilities for faster computation.

3.7.9 Limitations and Considerations

- GPU Memory Constraints: GPU memory is typically more limited than CPU memory, and developers need to manage memory efficiently.
- Data Transfer Overhead: Transferring data between the CPU and GPU incurs overhead, and minimizing data transfer is crucial for performance.

3.7.10 CUDA Community and Resources

- **Developer Community:** The CUDA community is active and provides forums and resources for developers to collaborate and seek assistance.
- **Documentation and Tutorials:** NVIDIA offers comprehensive documentation and tutorials to help developers get started with CUDA programming.

CUDA has played a significant role in advancing the field of parallel computing and enabling GPU acceleration for a wide range of applications. It has become a key technology for researchers and developers working on computationally intensive tasks, offering a powerful toolset for harnessing the parallel processing capabilities of NVIDIA GPUs.

3.8 Anaconda

Anaconda is an open-source distribution and platform that has emerged as a cornerstone in the fields of data science and machine learning. More than just a package manager, Anaconda provides a comprehensive ecosystem that includes Python (or R), popular libraries, and tools crucial for scientific computing. At its core is Conda, a powerful package manager simplifying the installation, management, and updating of libraries and dependencies. Anaconda Navigator, with its graphical user interface, facilitates seamless environment and package management, enhancing user experience. The flexibility of creating isolated environments ensures project-specific dependencies, fostering reproducibility and avoiding conflicts. This distribution comes bundled with Jupyter Notebooks, fostering interactive coding and documentation. The strength of Anaconda lies not only in its package management capabilities but also in its curated selection of data science and machine learning libraries such as Pandas, NumPy, Scikit-learn, TensorFlow, and more. It has become a go-to solution for individuals and enterprises, supported by an active community, extensive documentation, and initiatives promoting its educational use. With integration capabilities with popular IDEs and cloud services, Anaconda offers a versatile and robust environment for tackling diverse data-centric challenges. Figure 3.14 shows the interface of Anaconda navigator.



Figure 3.14 : Anaconda navigator

3.8.1. Overview

- **Distribution:** Anaconda is more than just a package manager; it is a distribution that includes Python (or R), popular libraries, and tools for data science and machine learning.
- **Package Manager:** Conda, the package manager used by Anaconda, simplifies the process of installing, managing, and updating libraries and dependencies.

3.8.2 Key Components

Conda: Conda is the package manager that comes with Anaconda. It handles package management and dependency resolution, making it easy to install, update, and remove packages.

Anaconda Navigator: Anaconda Navigator is a graphical user interface (GUI) that simplifies the management of environments, packages, and channels.

Jupyter Notebooks: Jupyter Notebooks are included in Anaconda, providing an interactive environment for developing, testing, and documenting code. It supports various programming languages, including Python, R, and Julia.

3.8.3 Environments

Isolation: Anaconda allows the creation of isolated environments, each with its own set of packages and dependencies. This helps avoid conflicts between different projects and ensures reproducibility.

Environment Files: Environment specifications can be saved to a YAML file (environment.yml) for easy sharing and replication of environments.

yaml code

name: myenv

channels:

- defaults

- conda

- forge

dependencies:

- python=3.8

- numpy

- pandas

- scikit-learn

3.8.4 Package Management

Installing Packages: Conda makes it easy to install packages and their dependencies. For example:

conda install numpy

Updating Packages: Packages can be updated with a simple command:

conda update numpy

Virtual Environments: Virtual environments can be created and activated with:

conda create --name myenv conda activate myenv

3.8.5 Data Science Libraries

Pandas: A powerful library for data manipulation and analysis.

NumPy: Provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on them.

Matplotlib and Seaborn: Libraries for creating static, animated, and interactive visualizations in Python.

SciPy: An open-source library for mathematics, science, and engineering.

3.8.6 Machine Learning Libraries

Scikit-learn: A simple and efficient tool for data analysis and modeling.

TensorFlow and PyTorch: Deep learning frameworks widely used for building and training neural networks.

XGBoost and LightGBM: Libraries for gradient boosting algorithms, commonly used in machine learning competitions.

3.8.7 Interoperability

Integration with IDEs: Anaconda integrates seamlessly with popular integrated development environments (IDEs) like VSCode, JupyterLab, and Spyder.

3.8.8 Community and Support

Active Community: Anaconda has a large and active community of users and developers. Support is available through forums, documentation, and community-driven resources.

3.8.9 Education and Training

Educational Initiatives: Anaconda is widely used in educational settings, and there are initiatives to promote its use in teaching data science and programming.

Training Resources: Numerous tutorials, courses, and documentation are available to help users get started and become proficient with Anaconda.

3.8.10 Deployment and Scaling

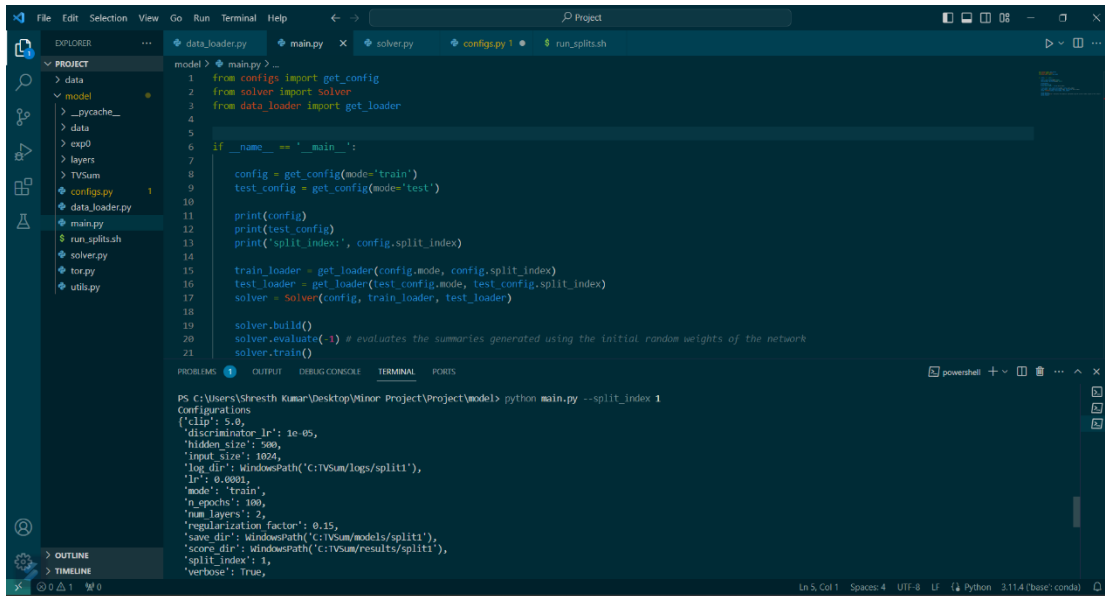
- Enterprise Solutions: Anaconda provides enterprise solutions for managing and deploying data science and machine learning applications at scale.
- Cloud Integration: Anaconda can be seamlessly integrated with cloud services, allowing users to deploy and scale their applications in cloud environments.

Anaconda has become a go-to solution for data scientists, researchers, and developers working in the fields of data science and machine learning. Its comprehensive suite of tools, libraries, and the ease of package management make it a versatile platform for developing and deploying data-driven applications. Whether you are a beginner or an experienced practitioner, Anaconda provides a robust and flexible environment for tackling a wide range of data-related tasks.

Chapter 4 – Results

The Results chapter is the heart of our project report, showcasing the key findings from our research efforts. This section is all about presenting the data we've collected and analyzed, using graphs and tables to make it easy to understand.

Figure 4.1 shows the training steps using splits.

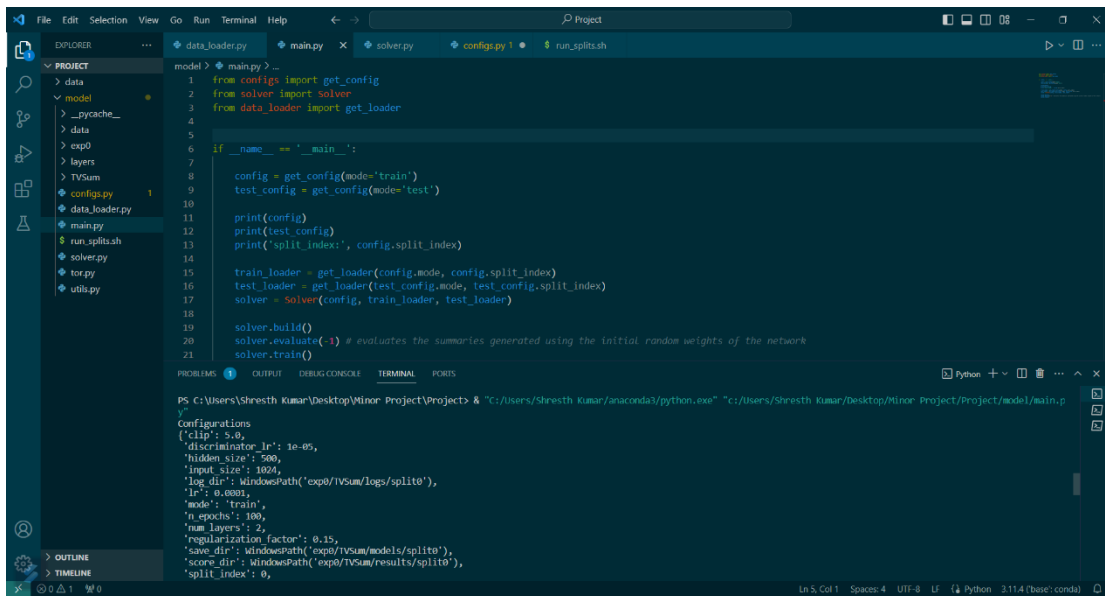


```
1 from configs import get_config
2 from solver import solver
3 from data_loader import get_loader
4
5
6 if __name__ == '__main__':
7
8     config = get_config(mode='train')
9     test_config = get_config(mode='test')
10
11     print(config)
12     print(test_config)
13     print('split_index:', config.split_index)
14
15     train_loader = get_loader(config.mode, config.split_index)
16     test_loader = get_loader(test_config.mode, test_config.split_index)
17     solver = Solver(config, train_loader, test_loader)
18
19     solver.build()
20     solver.evaluate(-1) # evaluates the summaries generated using the initial random weights of the network
21     solver.train()
```

```
PS C:\Users\Shreshth Kumar\Desktop\Minor Project\Project\model> python main.py --split_index 1
Configurations
{'clip': 5.0,
 'discriminator_lr': 1e-05,
 'hidden_size': 500,
 'input_size': 1024,
 'log_dir': 'WindowsPath("C:/TVSum/logs/split1")',
 'lr': 0.0001,
 'mode': 'train',
 'n_epochs': 100,
 'num_layers': 2,
 'regularization_factor': 0.15,
 'save_dir': 'WindowsPath("C:/TVSum/models/split1")',
 'score_dir': 'WindowsPath("C:/TVSum/results/split1")',
 'split_index': 1,
 'verbose': True}
```

Figure 4.1: Training Step using Splits

Figure 4.1 shows the training steps using split 0.



```
1 from configs import get_config
2 from solver import solver
3 from data_loader import get_loader
4
5
6 if __name__ == '__main__':
7
8     config = get_config(mode='train')
9     test_config = get_config(mode='test')
10
11     print(config)
12     print(test_config)
13     print('split_index:', config.split_index)
14
15     train_loader = get_loader(config.mode, config.split_index)
16     test_loader = get_loader(test_config.mode, test_config.split_index)
17     solver = Solver(config, train_loader, test_loader)
18
19     solver.build()
20     solver.evaluate(-1) # evaluates the summaries generated using the initial random weights of the network
21     solver.train()
```

```
PS C:\Users\Shreshth Kumar\Desktop\Minor Project\Project> & "C:/Users/Shreshth Kumar/anaconda3/python.exe" "C:/Users/Shreshth Kumar/Desktop/Minor Project/Project/model/main.p
Configurations
{'clip': 5.0,
 'discriminator_lr': 1e-05,
 'hidden_size': 500,
 'input_size': 1024,
 'log_dir': 'WindowsPath("exp0/TVSum/logs/split0")',
 'lr': 0.0001,
 'mode': 'train',
 'n_epochs': 100,
 'num_layers': 2,
 'regularization_factor': 0.15,
 'save_dir': 'WindowsPath("exp0/TVSum/models/split0")',
 'score_dir': 'WindowsPath("exp0/TVSum/results/split0")',
 'split_index': 0}
```

Figure 4.2: Training Step using Split 0

Figure 4.3 shows the loss curves of the discriminator and generator

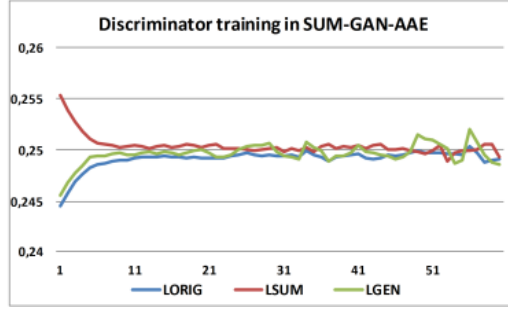


Figure 4.3: Loss curves of the discriminator (LORIG, LSUM) and generator (LGEN) of SUM-GAN-AAE model.

Performance (FScore (%)) of SUM-GANAAE for different values of the regularization factor if given in table 4.1.

Table 4.1: Performance (FScore (%)) of SUM-GANAAE for different values of the regularization factor.

	SumMe	TVSum
$\sigma = 0.05$	47.1	58.3
$\sigma = 0.1$	48.2	58.2
$\sigma = 0.15$	48.9	58.3
$\sigma = 0.3$	47.6	57.3
$\sigma = 0.5$	46.8	59.6

Comparison (F-Score (%)) with different unsupervised video summarization approaches, on SumMe and TVSum is given in table 4.2.

Table 4.2: Comparison (F-Score (%)) with different unsupervised video summarization approaches, on SumMe and TVSum.

	SumMe	TVSum
Random summary	39.9 (–)	53.9 (–)
Tessellation [14]	41.4 (–)	64.1 (+)
DR-DSN [32]	41.4 (–)	57.6 (–)
Online Motion-AE [29]	37.7 (–)	51.5 (–)
UnpairedVSN [20]	47.5 (–)	55.6 (–)
SUM-GAN-sl [1]	47.3 (–)	58.0 (–)
SUM-GAN-VAAE	45.7 (–)	57.6 (–)
SUM-GAN-AAE	48.9	58.3

Comparison (F-Score (%)) of our unsupervised method with supervised video summarization approaches on SumMe and TVSum is shown in table 4.3.

Table 4.3: Comparison (F-Score (%)) of our unsupervised method with supervised video summarization approaches on SumMe and TVSum.

	SumMe	TVSum		SumMe	TVSum
Random summary	39.9 (−)	53.9 (−)	MAVS [8]	40.3 (−)	66.8 (+)
vsLSTM [27]	37.6 (−)	54.2 (−)	SUM-FCN [21]	47.5 (−)	56.8 (−)
dppLSTM [27]	38.6 (−)	54.7 (−)	SUM-DeepLab [21]	48.8 (−)	58.4 (+)
H-RNN [30]	41.1 (−)	57.7 (−)	DR-DSNsup [32]	42.1 (−)	58.1 (−)
Tessellationsup [14]	37.2 (−)	63.4 (+)	ActionRanking [6]	40.1 (−)	56.3 (−)
HSA-RNN [31]	44.1 (−)	59.8 (+)	UnpairedVSNpsup [20]	48.0 (−)	56.1 (−)
DQSN [33]	−	58.6 (+)	VASNet [7]	49.7 (+)	61.4 (+)
DSSE [26]	−	57.0 (−)	SUM-GAN-AAE	48.9	58.3

Comparison (FScore (%)) of the best performing SUM-GAN model (based on [16]) with the proposed model for different values of the regularization term σ is given in table 4.4 .

Table 4.4: Comparison (FScore (%)) of the best performing SUM-GAN model (based on [16]) with the proposed model for different values of the regularization term σ .

	σ	SumMe	TVSum
SUM-GAN	0.3	38.7	50.8
	0.05	53.6	60.9
SUM-GAN-AAE	0.1	55.4	59.9
	0.15	56.4	60.2
	0.3	56.0	59.8
	0.5	56.9	63.9

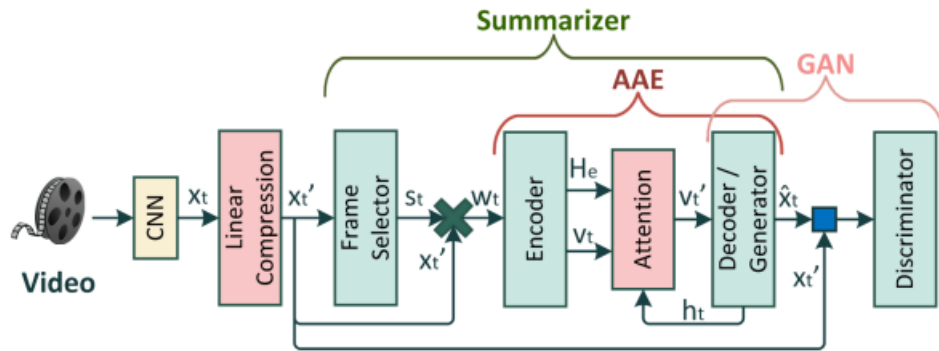


Figure 4.4: The proposed SUM-GAN-AAE architecture

Figure 4.4 shows the proposed SUM-GAN-AAE architecture. Our unsupervised SUM-GAN-AAE model was compared also against supervised methods for video summarization (which is a rather unfair comparison). Table 4.3 shows that: i) the best methods on TVSum are highly-adapted to this dataset as they exhibit random-level performance on SumMe; ii) only a few supervised methods surpass the performance of a random summary generator on

both datasets. The performance of these methods ranges in 44.1 – 49.7 on SumMe, and in 56.1 – 61.4 on TVSum. Hence, the results of our unsupervised method make SUM-GAN-AAE comparable with SoA supervised algorithms.

Chapter 5 – Conclusion and future Scope

5.1 Conclusion

In conclusion, this project represents a pioneering endeavour at the intersection of artificial intelligence and video analytics. By leveraging the power of Generative Adversarial Networks (GANs), the project aims to transform traditional video summarization methods, introducing an innovative approach that prioritizes attention-driven learning and discriminative adversarial training. The fusion of attention mechanisms allows the model to dynamically emphasize crucial segments within a video, enabling a more nuanced understanding of contextual significance. The adversarial learning component introduces a competitive edge, challenging the model to generate summaries that not only capture salient features but also align closely with human-created summaries.

Through the integration of attention-driven mechanisms and adversarial learning, the project seeks to address the limitations of existing video summarization techniques. The anticipated outcomes include enhanced summarization accuracy, improved contextual coherence, and adaptability to diverse content genres. The project's significance lies in its potential to set a new standard for video summarization, offering a model capable of discerning intricate details and relationships within videos that may elude traditional methods.

As we move forward, the outcomes of this research hold promise for revolutionizing the way we process and comprehend vast amounts of video content. The implications extend to diverse applications, including video analysis, content retrieval, and information comprehension across various domains. By pushing the boundaries of video summarization, this project contributes to the advancement of artificial intelligence and reinforces the notion that innovative approaches, such as attention-driven adversarial learning, can redefine the landscape of video analytics.

5.2 Scope of Future

The future scope of a GAN-based video summarization project is promising and holds potential for several advancements and applications. Here are some key areas of future exploration and development:

Refinement of Attention Mechanisms: Further research and development can focus on refining attention mechanisms within the GAN framework. Optimizing how the model

allocates attention to different regions of a video could lead to more nuanced and contextually relevant summaries.

Enhanced Adversarial Training Strategies: Investigating and implementing advanced adversarial training strategies can contribute to the robustness and generalization of the model. This includes exploring novel loss functions, training methodologies, and techniques to handle diverse video content.

Multi-Modal Video Summarization: Extending the project to support multi-modal summarization, involving not only visual but also textual and auditory modalities, could enhance the overall understanding and representation of complex videos, making the system more versatile.

Real-Time Summarization: Advancing the project to achieve real-time video summarization capabilities would be valuable for applications that require immediate insights, such as live streaming, surveillance, or event monitoring.

Large-Scale Datasets and Benchmarking: Expanding the project's scope to accommodate and perform well on larger and more diverse datasets can contribute to its applicability in real-world scenarios. Benchmarking against state-of-the-art methods will help assess the model's comparative performance.

User-Centric Customization: Introducing features that allow users to customize and guide the summarization process based on specific preferences or requirements could enhance the usability and adoption of the system in various domains.

Deployment in Industry Verticals: Exploring the deployment of the GAN-based video summarization in specific industry verticals, such as healthcare, education, or entertainment, can lead to tailored solutions that address domain-specific challenges and requirements.

Explainability and Interpretability: Incorporating mechanisms for explaining and interpreting the summarization decisions made by the model can enhance trust and transparency, making the technology more accessible and acceptable in sensitive applications.

Collaborative Filtering for Summarization: Investigating collaborative filtering techniques where the model learns from user feedback on summaries can result in adaptive systems that continually improve based on user preferences.

Integration with Autonomous Systems: Integrating GAN-based video summarization into autonomous systems, such as self-driving cars or drones, can contribute to improved perception and decision-making capabilities in dynamic environments.

Ethical Considerations and Bias Mitigation: Proactively addressing ethical considerations, including biases in the training data and potential implications of summarization decisions, is crucial for responsible AI development.

As the field of video summarization continues to evolve, the future scope lies in pushing the boundaries of technology, making these systems more intelligent, adaptable, and ethical for diverse applications across industries.

REFERENCES

- [1] P. Saini, K. Kumar, S. Kashid, A. Saini, and A. Negi, "Video summarization using Deep Learning Techniques: A detailed analysis and investigation," *Artificial Intelligence Review*, 2023. doi:10.1007/s10462-023-10444-0
- [2] X. Wang, Y. Li, H. Wang, L. Huang, and S. Ding, "A video summarization model based on deep reinforcement learning with long-term dependency," *Sensors*, vol. 22, no. 19, p. 7689, 2022. doi:10.3390/s22197689
- [3] H. B. Ul Haq, M. Asif, M. B. Ahmad, R. Ashraf, and T. Mahmood, "An effective video summarization framework based on the object of interest using Deep Learning," *Mathematical Problems in Engineering*, vol. 2022, pp. 1–25, 2022. doi:10.1155/2022/7453744
- [4] A. Negi, K. Kumar, and P. Saini, "Object of interest and unsupervised learning-based framework for an effective video summarization using Deep Learning," *IETE Journal of Research*, pp. 1–12, 2023. doi:10.1080/03772063.2023.2220693
- [5] E. Apostolidis, E. Adamantidou, A. I. Metsai, V. Mezaris, and I. Patras, "Unsupervised video summarization via attention-driven adversarial learning," *MultiMedia Modeling*, pp. 492–504, 2019. doi:10.1007/978-3-030-37731-1_40
- [6] M. Basavarajaiah and P. Sharma, "Survey of compressed domain video summarization techniques," *ACM Computing Surveys*, vol. 52, no. 6, pp. 1–29, 2019. doi:10.1145/3355398
- [7] M. Basavarajaiah and P. Sharma, "GVSUM: Generic video summarization using Deep Visual Features," *Multimedia Tools and Applications*, vol. 80, no. 9, pp. 14459–14476, 2021. doi:10.1007/s11042-020-10460-0
- [8] J. Fajtl, H. S. Sokeh, V. Argyriou, D. Monekosso, and P. Remagnino, "Summarizing videos with attention," *Computer Vision – ACCV 2018 Workshops*, pp. 39–54, 2019. doi:10.1007/978-3-030-21074-8_4
- [9] E. Apostolidis, E. Adamantidou, A. I. Metsai, V. Mezaris, and I. Patras, "Video summarization using Deep Neural Networks: A survey," *Proceedings of the IEEE*, vol. 109, no. 11, pp. 1838–1863, 2021. doi:10.1109/jproc.2021.3117472
- [10] S. Arslan and S. Tanberk, "Key frame extraction with attention based deep neural networks," arXiv.org, <https://arxiv.org/abs/2306.13176> (accessed Sep. 30, 2023).

- [11] H. Huang and R. K. Wong, “Attention-based SEQ2SEQ regularisation for relation extraction,” *2021 International Joint Conference on Neural Networks (IJCNN)*, 2021. doi:10.1109/ijcnn52387.2021.9533807
- [12] Li, P.; Tang, C.; Xu, X. Video summarization with a graph convolutional attention network. *Front. Inf. Technol. Electron. Eng.* 22, 902-913
- [13] Jadon, S.; Jasim, M. Unsupervised video summarization framework using keyframe extraction and video skimming. In *Proceedings of the 2020 IEEE 5th International Conference on Computing Communication and Automation (ICCCA)*, Greater Noida, India, 30–31 October 2020; pp. 140–145.
- [14] Gygli, M.; Grabner, H.; Riemenschneider, H.; Gool, L.V. Creating summaries from user videos. In *European conference on computer vision*, Switzerland, 2014; pp. 505–520.
- [15] Zhou, K.; Qiao, Y.; Xiang, T. Deep reinforcement learning for unsupervised video summarization with diversity-representativeness reward. In *Proceedings of the AAAI Conference on Artificial Intelligence*, New Orleans, LA, USA, 2–7 February 2018.
- [16] Yaliniz, G.; Ikizler-Cinbis, N. Unsupervised Video Summarization with Independently Recurrent Neural Networks. In *Proceedings of the 2019 27th Signal Processing and Communications Applications Conference (SIU)*, Sivas, Turkey, 24–26 April 2019.
- [17] Liang, G.; Lv, Y.; Li, S.; Zhang, S.; Zhang, Y. Unsupervised Video Summarization with a Convolutional Attentive Adversarial Network. *arXiv* 2021, arXiv:2105.11131.
- [18] Apostolidis, E.; Adamantidou, E.; Metsai, A.I.; Mezaris, V.; Patras, I. AC-SUM-GAN: Connecting actor-critic and generative adversarial networks for unsupervised video summarization. *IEEE Trans. Circuits Syst. Video Technol.* 2020, 31, 3278–3292.
- [19] Fu, T.-J.; Tai, S.-H.; Chen, H.-T. Attentive and adversarial learning for video summarization. In *Proceedings of the 2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Waikoloa, HI, USA, 7–11 January 2019; pp. 1579–1587.
- [20] Apostolidis, E., et al.: A stepwise, label-based approach for improving the adversarial training in unsupervised video summarization. In: *AI4TV, ACM MM* 2019
- [21] Bahuleyan, H., et al.: Variational attention for sequence-to-sequence models. In: *27th COLING*. pp. 1672–1682 (2018)

- [22] Elfeki, M., et al.: Video summarization via actionness ranking. In: IEEE WACV 2019. pp. 754–763
- [23] Feng, L., et al.: Extractive video summarizer with memory augmented neural networks. In: ACM MM 2018. pp. 976–9
- [24] Fu, T., et al.: Attentive and adversarial learning for video summarization. In: IEEE WACV 2019. pp. 1579–1587
- [25] Gygli, M., et al.: Creating summaries from user videos. In: ECCV 2014. pp. 505–520
- [26] Ji, Z., et al.: Video summarization with attention-based encoder-decoder networks. IEEE Trans. on Circuits and Systems for Video Technology pp. 1–1 (2019)