University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Data Science

Self-Supervised Learning: Video Clip Order Prediction with diffusion models

Supervisor
Professor Lamberto Ballan
Co-supervisors
Alessio Del Bue and Gianluca Scarpellini

Master Candidate
Sara Repetto

Academic Year 2022-2023

Abstract

In recent years, neural networks have achieved incredible performance in computer vision applications like image classification and video recognition due to the availability of powerful computational resources and vast amounts of data.

Though a large amount of data exists, it takes great effort to annotate such massive data, which is necessary for standard supervision tasks. To reduce the need for annotation, a self-supervised approach can be used.

In the self-supervised approach, some surrogate tasks that don't need the use of labels data are used to learn a data representation. This approach can be useful to extract meaningful insight from data that can then be exploited in different downstream tasks.

With this work, we propose to use clip order prediction as a self-supervised task. Specifically, each video is divided into clips, with the same number of frames for each clip. These clips are then shuffled, and the objective of the task is to rearrange them in the correct order.
The usefulness of the learned representation is tested in different downstream tasks. The learned representation is used to perform an action classification task at the video level. Additionally, the efficacy of the acquired representation was assessed through a frame-level phase classification task.

"Life is really simple, but we insist on making it complicated"

— Confucius

Acknowledgements

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Lamberto Ballan, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

Vorrei ringraziare Gianluca per la pazienza e il supporto fornito durante lo svolgimento degli esperimenti.

Desidero ringraziare con affetto la mia famiglia per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.

Ho desiderio di ringraziare poi i miei amici per tutti i bellissimi anni passati insieme e le mille avventure vissute.

Padova, December 2023 Sara Repetto

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The supervised learning approach is the most used approach in machine learning. A supervised learning approach is a machine learning paradigm in which an algorithm is trained on a labeled dataset to make predictions or classifications.
The algorithm can map input data to the appropriate output or target variable in supervised learning by utilizing labeled examples in the training data.
Supervised learning aims to construct a model capable of extrapolating relationships and patterns from the training data to generate precise predictions on novel, unobserved data. The algorithm is trained using labeled data to acquire knowledge of the underlying patterns and relationships between the input features and the target variable. During training, the model iteratively updates its internal parameters to minimize discrepancies between predicted outputs and the true labels.
One of the challenges in supervised learning is to prevent overfitting, where the model learns to fit the training data too closely and performs poorly on new data. To reduce the risk of overfitting, a considerable amount of data is needed to ensure enough variability.

In recent years, neural networks have achieved incredible results in the supervised learning approach. However, considerable data must be used to achieve these incredible results. Though a large amount of data exists, annotating such massive data requires great effort, which is necessary in standard supervision tasks.

Self-supervised learning is another machine learning technique that is gaining popularity because of its ability to avoid the cost of annotating large-scale datasets. Indeed, self-supervised learning data are not annotated, and information already contained in the data is exploited to learn meaningful representations of the data.
Self-supervised learning creates pretext tasks or objectives that encourage the model to capture meaningful features from the data. These pretext tasks are designed to be easy for the model to learn, leveraging inherent data structure.
The learned representations can be used for several downstream tasks by exploiting transfer learning. Transfer learning is a machine learning technique where a model trained on one task is adapted for related tasks instead of starting a model's training from scratch. Therefore, transfer learning leverages the knowledge gained from a source task to improve performance on a target task.
Using a self-supervised learned representation to perform some downstream tasks makes these tasks easier because a meaningful representation is already provided and

thus reduces the amount of data and consequent annotations needed.

In this work, a self-supervised task is used to learn a meaningful representation of the data.
As a self-supervised method, a video clip order prediction task is used. In this task, the video is divided into several clips. The clips are then shuffled, and the goal is to predict the correct ordering of these clips. This task is self-supervised because the correct position of the clips is easily obtained directly from the data without any additional annotation.
Using the clip order prediction task, the model is forced to learn a representation of the data to predict the original position of each clip correctly. This task exploits the semantic coherence of clips within videos, so clips closer temporally should have a more similar representation.
A backbone was used to extract features from the clips. Then, a diffusion model is employed to determine each clip's position and, thus, order. Specifically, a diffusion formulation is used to add some noise to the correct position of clips repeatedly. During the train, the model learns to reverse the process to recover the proper locations.
Then, the learned representation is exploited in different downstream tasks. The backbone trained on the reordering task is finetuned on an action classification task. The same method is used to test the efficiency of the learned representation in a phase classification task.

1.1 Thesis outline

The remaining part of the thesis will be structured as follows:

- **Chapter 2**: brief introduction to the supervised learning approach. The most frequently used self-supervised tasks in computer vision are presented, and a brief literature review is given.
- **Chapter 3**: brief introduction to generative and diffusion probabilistic models. The functioning of diffusion models is described in detail, and the main works on computer vision are presented.
- **Chapter 4**: brief introduction to graphs and graph neural networks. The different types of graphs are described in detail, and the main graph neural networks are described.
- **Chapter 5**: description of the proposed approach. This chapter describes in detail the methodology used in the proposed approach. In particular, the type of input used, the literature review, and the model used are described.
- **Chapter 6**: description of the experiments. In this chapter, the experiments performed are described in detail. Therefore, it describes how the reordering task is performed and how the learned representation is tested in different downstream tasks.
- **Chapter 7**: conclusion and future directions. In this chapter, the results obtained are analyzed, and other possible topics for investigation are proposed.

Chapter 2

Self-supervised learning approach

Self-supervised learning is a machine learning method where a model learns to predict outcomes for some input data without explicit labeling from humans. Instead of relying on annotated data, self-supervised learning leverages the inherent structure or information within the data to create supervision signals.
Self-supervised learning uses some tasks based only on intrinsic data characteristics to encourage a model to learn about the underlying structure of the data. As the model completes the auxiliary task, it learns a meaningful representation for different downstream tasks.
The self-supervised learning approach has grown in recent years due to its potential to address some critical challenges in machine learning and artificial intelligence.
First, self-supervised learning reduces reliance on large amounts of labeled data. This is important because labeling data, particularly for specialist or emerging topics, can be time-consuming, expensive, and occasionally impossible. Self-supervised learning can efficiently use available resources using self-generated labels from unlabeled data.
Furthermore, models pre-trained using self-supervised techniques tend to learn rich, general-purpose data representations. Therefore, self-supervised learning can be a practical pretraining step for various downstream tasks. These learned representations can be fine-tuned on specific tasks, leading to improved performance, faster convergence, and reduced data requirements for task-specific training.
Several self-supervised methods have been developed in recent years.

2.1 Contrastive learning-based self-supervised learning methods

Contrastive learning is a machine learning methodology employed to acquire significant data representations. It finds widespread application in self-supervised learning, specifically in computer vision and natural language processing.
The main idea behind contrastive learning is to push similar data points closer in the embedding space while pushing dissimilar points farther apart. In this way, the model learns high-level attributes common between data classes and can help to set apart a data class from another. Similar data points are called positive pairs, while dissimilar

ones are called negative pairs.
Supervised methods require human annotations for each input sample x. In contrast, contrastive learning approaches solely necessitate the specification of a similarity distribution for sampling positive pairs and a dissimilarity distribution for sampling negative pairs.
Different methods of contrastive learning use different strategies to define those distributions.

2.1.1 Instance Discrimination Method

This approach uses data augmentation techniques to create two or more augmented views of the same data instance to obtain positive pairs. Depending on the domain, these augmentations can include cropping, rotation, color jittering, or other transformations. Negative pairs are commonly generated by randomly pairing data points within the dataset or by deliberately picking data points that are recognized to possess dissimilar characteristics. In general, in computer vision, to obtain negative pairs, an image is combined with another image that has been randomly selected from a distinct category. Figure 2.1 shows the instance discrimination method in detail. Individual views of the data are processed independently by a neural network (an encoder) to create an embedding for each view. Typically, these embeddings are lower-dimensional representations that effectively encapsulate significant information about the data.
The objective is to ensure that positive samples are embedded closer to each other, and the negative samples are pushed farther apart.
A contrastive loss is employed to enhance the similarity between positive pairs and the dissimilarity between negative pairs. A loss function commonly used is the triplet loss. This loss encourages the distance between a positive pair to be smaller than the distance between a negative pair by a certain margin:

$$\mathcal{L}(A, P, N) = \max\left(\|\mathrm{f}(A) - \mathrm{f}(P)\|_2 - \|\mathrm{f}(A) - \mathrm{f}(N)\|_2 + \alpha, 0\right)$$

where A is an anchor input, P is a positive input of the same class as A, N is a negative input of a different class from A, α is a margin between positive and negative pairs, and f is an embedding.

2.1.2 Cluster-based contrastive learning

Clustering-based contrastive learning approach integrates concepts from clustering and contrastive learning methodologies.
A clustering method is employed to establish positive and negative pairs. (16). The main objective is to promote the proximity of data points within the same cluster in the acquired embedding space while increasing the distance between data points originating from distinct clusters.
In the first step, an unsupervised clustering algorithm, such as K-means, DBSCAN, or hierarchical clustering, is employed to partition the unlabelled dataset into distinct groups based on the similarity of data points.
Each cluster is indicative of a collection of data points that are hypothesized to possess semantic or conceptual interconnections.
Once the data is clustered, an encoder is used to compute embeddings so that points that belong to the same cluster are pushed together, and points that belong to different clusters are pulled apart. In this context, the clusters serve as pseudo-labels for the data points within them.

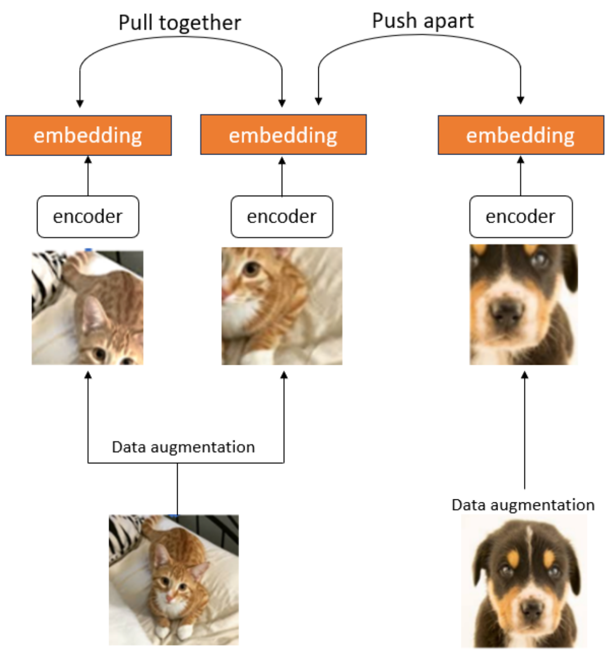


Figura 2.1: Instance discrimination method

2.2 Generative models

In generative modeling, the goal is to reconstruct the original input while learning a meaningful latent representation.
The most promising generative models are autoencoders (AEs) and generative adversarial nets (GANs).

2.2.1 Autoencoders(AEs)

Autoencoder was first introduced by Ballard et al.(17) as a method for unsupervised pretraining.
The term autoencoder refers to a feed-forward neural network that produces its input at the output layer.
As described in figure2.2, an autoencoder is constituted by an encoder network and a decoder network. The encoder network maps the input into a lower dimensional space. The decoder instead reconstructs the input from the latent representation in the lower dimensional space.
Generally, a mean-square error loss is used to force the reconstruction to be as close as possible to the input.
Reconstructing the input from a representation with smaller dimensions than the input forces the model to focus on the most salient aspects for reconstruction, thus learning a more compact and informative representation of the source data.

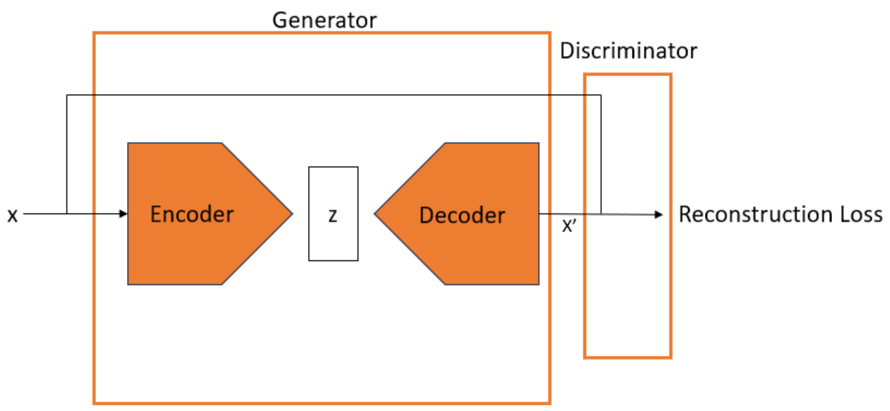


Figura 2.2: Autoencoder network

2.2.2 Generative adversarial nets (GANs)

The generative adversarial net was first introduced by Ian Goodfellow et al .(18).
As described in figure 2.3 a GAN consists of two neural networks: a generator and a discriminator.
The network generator takes random noise as input and generates data samples. Its

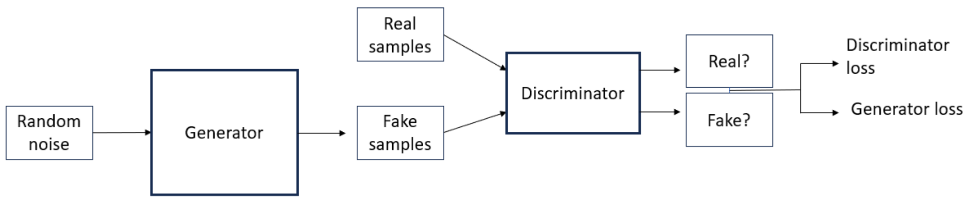


Figura 2.3: Generative Adversarial Network (GAN)

purpose is to generate data that cannot be distinguished from actual data.
The discriminator network assigns a probability score to each sample, indicating whether it believes the sample to be genuine or fake, based on real data from the training set and data generated by the generator.
The objective of the discriminator is to become better at distinguishing between real and generated data. Instead, the generator has to learn to create examples as similar as possible to the original input. Creating such examples from random noise during training gives it a meaningful representation of the input.

2.3 Spatial Context Prediction

Predicting missing or contextual information has been one of the most popular unsupervised learning techniques.

Several context prediction tasks have been developed in computer vision.

2.3.1 Colorization

Colorization aims to restore full information about the color of each pixel from a grey-scale image. The process of colorization appears to be challenging initially since most of the information is lost. In many instances, however, the semantics of the scene and its surface texture provide sufficient signals for many regions in each image.
The training data for color prediction is practically free: any color image can be used as a training example by converting it into a grey-scale image and using the original image as a supervisory signal.
In the work of Zhang et al. (1), a CNN is trained to map from a grayscale input to a distribution over quantized color value outputs using the architecture shown in Figure 2.4. The proposed architecture is similar to an autoencoder, except that it employs

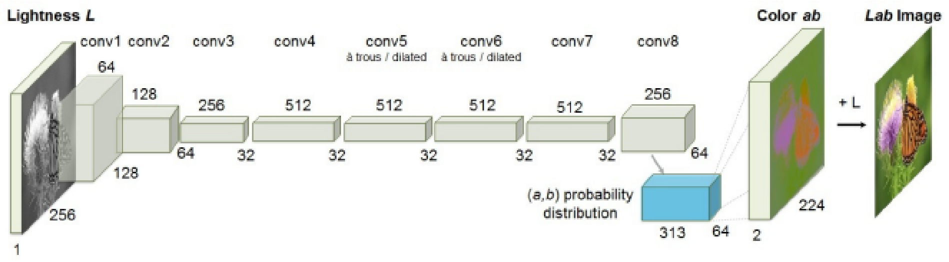


Figura 2.4: The CNN architecture taken from the work of Zhang et al.(1)

distinct image channels for input and output.
The experimental results in the work of Zhang et al. (1) have shown that learning feature representations via colorization as a pretext task is effective for solving object detection and segmentation problems; it is also more useful in image classification as compared to other self-supervised methods. Indeed, the learned representation can be used efficiently to train a linear classifier to perform object classification, detection, and segmentation.

2.3.2 Patches location

Another task commonly used as a self-supervised method in computer vision is patch location.
Patches location is a type of self-supervised learning that involves the model's ability to forecast the approximate location of picture patches. To accomplish this, the model has to acquire knowledge of spatial context, enabling it to discern the appropriate placement of these image patches.
Doersch et al. (2) used the relative spatial co-location of patches in images as a label for their self-supervised task.
Their model predicts the relative position between the central patch of an image and a second patch selected randomly from its eight neighboring patches (see Figure 2.5). Specifically, a vast assortment of unannotated images are used to extract random pairs of patches from each image. Each patch is fed into a convolutional neural network that follows the AlexNet architecture. A softmax layer outputs the probability of each

spatial configuration to predict the correct position.
The fundamental hypothesis is that achieving success in this task necessitates com-

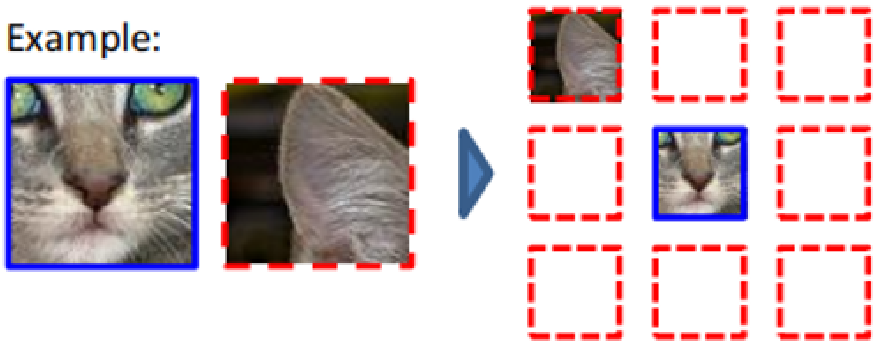


Figura 2.5: Patches reordering (Doersch et al. (2))

prehension of scenes and objects. Indeed, the task is much easier if the model has a global understanding of the scene .
The feature representation learned using this pre-text task has also proven useful for object detection, providing a significant boost on PASCAL VOC 2007 compared to learning from scratch. This means that the task helps to learn a meaningful, generalizable data representation.

Another task that exploits patch location information is the jigsaw puzzle task.
It was first introduced by Noroozi et al. in 2016 (19). This task involves solving puzzles where an image or object is divided into multiple irregularly shaped pieces, and the objective is to rearrange those pieces to recreate the original image or object.
To reorder the elements correctly, the model has to learn to extract the similarities and differences between the various elements of the puzzle, thus learning a meaningful representation of the scene.
In (19), it has been proven that the features learned are highly transferrable to detection and classification.

Finally, in the work of Camporese et al. (20), patch information is used to learn a meaningful representation of an image. Specifically, each image is divided into patches. Then Vision Transformer (ViT) is used to make predictions about the input patches. Different predictions are computed. First, the model has to predict the spatial relation class of pairs of image patches. Then, the model has to predict the distance between the input patch locations in the original image from pairs of image patches. Finally, the model is used to predict the 2-d location of the input image patch.

2.3.3 Image rotation

In their study, Gidaris et al.(3) randomly rotated an input image by one of four angles (0, 90, 180, or 270 degrees) and utilized CNN to estimate the accurate rotation via a four-class classification problem (see Figure 2.6).
To identify the rotation transformation implemented on an image, a model must possess a conceptual understanding of the objects depicted in the image, including

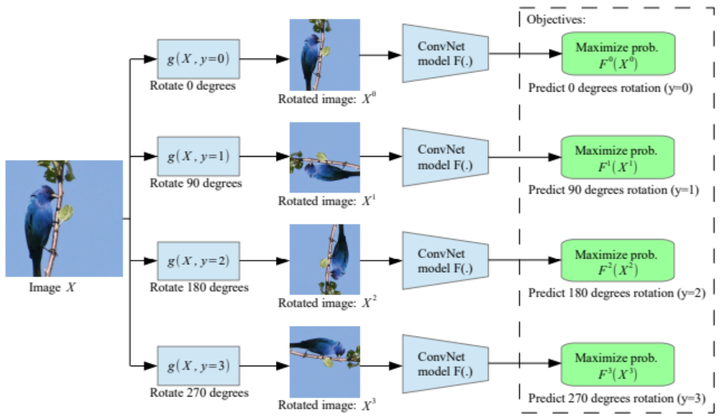

Objectives:
Image X
$g(X, y=0)$
Rotate 0 degrees
Rotated image: X^0
ConvNet model F(.)
Maximize prob. $F^0(X^0)$
Predict 0 degrees rotation (y=0)
$g(X, y=1)$
Rotate 90 degrees
Rotated image: X^1
Maximize prob. $F^1(X^1)$
Predict 90 degrees rotation (y=1)
$g(X, y=2)$
Rotate 180 degrees
Rotated image: X^2
Maximize prob. $F^2(X^2)$
Predict 180 degrees rotation (y=2)
$g(X, y=3)$
Rotate 270 degrees
Rotated image: X^3
Maximize prob. $F^3(X^3)$
Predict 270 degrees rotation (y=3)

Figura 2.6: Rotation task. Image taken from (3).

their pose, type, and location.
The experimental results provide evidence that this task effectively compels the model to acquire semantic features that are useful in different visual perception tasks, including object detection, object recognition, and object segmentation.

2.3.4 Learning to count

Noorozi et al.(21) describe a novel approach to representation learning, which involves the utilization of an artificial supervision signal that relies on counting visual primitives. This task is based on various assumptions. First, when an image is divided into distinct and non-overlapping regions, the cumulative count of visual primitives within each region must equate to the total count of primitives present in the original image. In addition, many forms of transformation, such as rotation or color jittering, do not alter the quantity of primitives in an image.
A convolutional neural network is used to count the number of visual primitives. This network is trained by exploiting the fact that the same image with different transformations must have the same number of primitives.
Experiments have shown that the learned representation appears to be useful in detection, classification, and segmentation tasks.

2.3.5 Predicting masked patches

Another task that uses spatial information is predicting missing image information. Trinh et al.(4), inspired by BERT(22), masked out a few patches in an image and tried to reconstruct the original image. The details of the proposed approach are shown in Figure 2.7.
The key idea is to use a portion of the input image to forecast the remaining portion of the image. Several square patches are initially sampled from the input. Depending on whether or not these patches are randomly selected to be masked out, they are subsequently routed into the encoder and decoder networks.

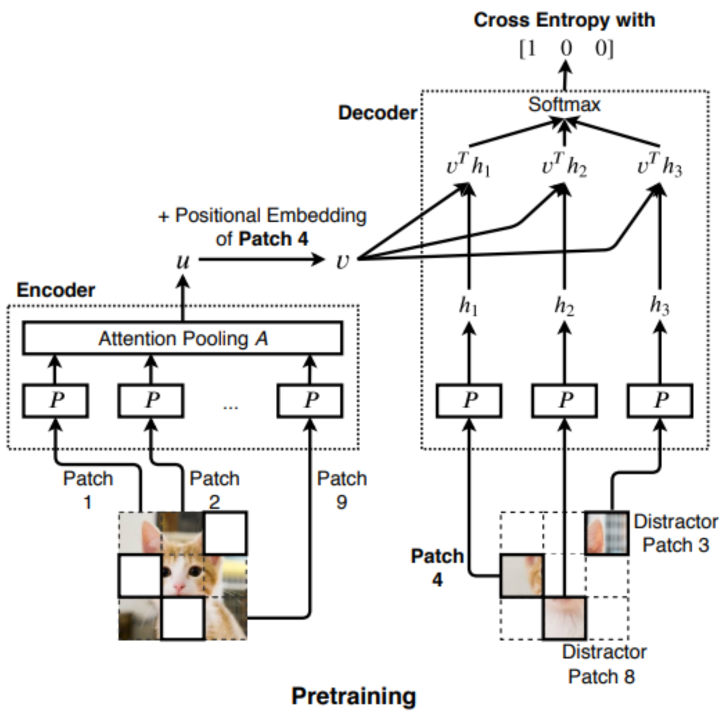


Figura 2.7: Predicting missing information task. Image taken from (4).

The same patch processing network P handles every patch. On the encoder side, the output vectors generated by P are fed into the attention pooling network to condense these representations into a single vector, u. On the decoder side, P generates the output vectors h1, h2, and h3.
The location embedding of a patch, chosen at random from the decoder's patches, is then added to the output vector u by the decoder to query the encoder, resulting in the creation of a vector v.
The similarity between each h and the vector v is then calculated using a dot product. After examining the dot products between v and h, the decoder must determine which patch should be used to fill in the place corresponding to the location embedding. This classification task employs the cross-entropy loss; the encoder and decoder are jointly trained using gradients back-propagation.

2.4 Self-supervised learning for videos

Different self-supervised tasks can be used to learn visual representation for videos. For videos, self-supervised tasks generally use temporal relations between frames as supervised signals. Indeed, the video comprises a series of frames that are semantically related. This suggests that frames in close temporal proximity exhibit a higher correlation, while frames that are far in time are less likely to exhibit a strong relationship

2.4.1 Frame order task

The frame order task is a self-supervised task that exploits the semantic relationship between frames. In the frame order task, the order of the frames is shuffled, and the model has to reorder the frames correctly.
Since frames that are close together are more similar than frames that are far apart in time, the model has to focus on similarities and differences among frames to perform this task correctly. In this way, the model learns a meaningful visual representation of the data. Furthermore, thinking about object transformations and relative placements across time is necessary for this reordering process. Thus, the representation is compelled to depict object appearances and deformations.

Misra et al.(23) employ a binary classification task to perform a temporal order verification. Temporal order verification involves making predictions regarding the validity of the entire sequence as opposed to individual elements within it. The authors investigate determining the "temporal validity" of a given sequence, specifically whether a given sequence of video frames follows the proper temporal order.
The frames of a video are shuffled, and the model receives both video with the correct sorting and video with shuffled frames as input. A Convolutional Neural Network (CNN) is employed to extract feature representation. The Convolutional Neural Network is employed on every individual frame inside the sequence and is trained "end-to-end" from a random initialization. Then, it is used to classify input videos as valid or invalid. The sequence verification task promotes the establishment of visual and temporal grounding in CNN features.
Empirical experiments demonstrated the usefulness of the learned representation. Indeed, compared to the initialization from scratch, pre-trained CNN showed improvements when fine-tuned in an action classification task and in a pose estimation task.

In the work of Lee et al.(5) a reordering task is performed. This work aims to train a neural network to sort a given tuple of randomly shuffled frames into their respective chronological order.
The main difference from the work of Misra et al.(23) is that the temporal order verification task is a binary classification task, whereas in the work of Lee et al.(5), a multi-class classification task is performed, where the aim is not only to predict the correctness or incorrectness of the sorting but the position of each frame.
In Figure 2.8, the main steps of the approach of Lee et al.(5) are shown. First of all, frames are sampled and shuffled. Then, features for each frame are encoded by convolutional layers. A Siamese architecture where all the branches share the same parameters was used. Then, a pairwise feature extraction on extracted features is performed. The final order prediction is then determined by concatenating all pairwise feature extractions after applying a softmax function and one fully connected layer.
The effectiveness of the learned representation was validated in different experimental conditions. Specifically, the ordering task was used successfully as an unsupervised pre-training approach to initialize models for action recognition, image classification, and object detection.

2.4.2 Clips order task

In the frame order task, the model learns to compare representations of frames that are very close in time. A similar approach can be used to have a more generalized

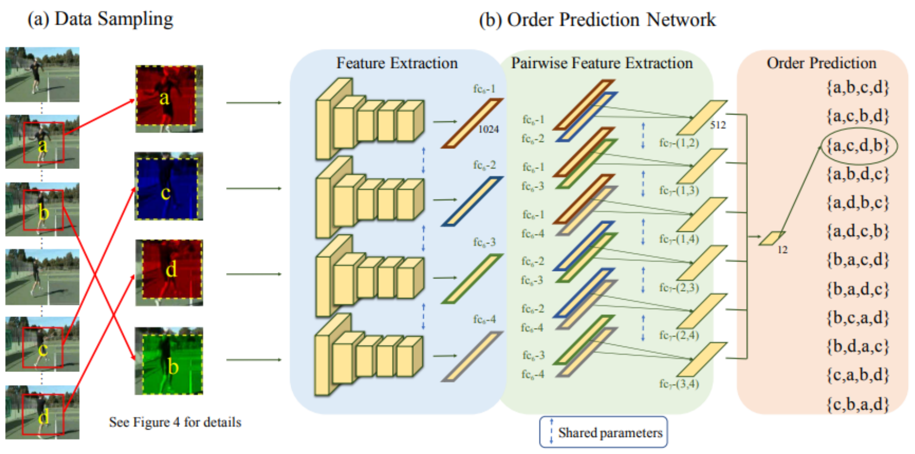


Figura 2.8: Overview of the approach of Lee at al.(5) . Image taken from (5)

representation that also allows comparison of more distant elements in time, but using clips to compare more temporally distant elements.
In addition, in the frame order task, the actual inputs are frames. Therefore, the learned models can only extract features from still images. Using clips instead of frames, the model learns a more complex representation of videos.
Lastly, the determination of the order is not unique when referring solely to frames. For example, given the arrangement of the frames shown in Figure 2.9, discerning the accurate sequence of frames proves challenging due to the potential viability of both orientations of the gymnast on the balance beam. Because of the internal dynamics

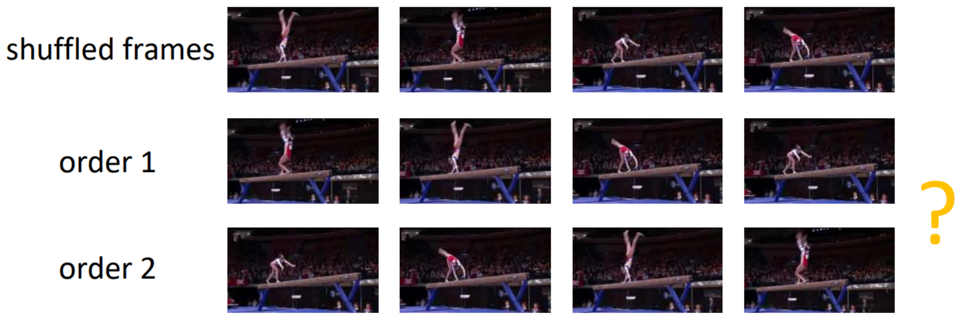


Figura 2.9: Example (Xu et al.(6))

included in each clip, the order will be more precise when clips are used in place of frames.
In the work of Xu et al. (6), different clips of 16 frames each are sampled from a video and then shuffled. 3D CNNs are used to extract features from each clip. Then, the order prediction task is resolved as a classification task. The features that have been extracted are combined and sent through two linear layers. Then, a SoftMax layer is applied to generate a probability distribution that represents the potential orders.
In the work of Xu et al.(6), the generalizability and usefulness of the learned represen-

tation has been demonstrated. Indeed, the learned 3D CNNs are used successfully as a pre-trained model for an action classification task.

2.4.3 Multi-view alignment

Another task commonly used as a self-supervised method in computer vision is view alignment.
View alignment task is used for data with multiple views of the same scene.
In this task, a latent representation is learned in such a way that frames taken from simultaneous viewpoints of the same video are encouraged to be close in the embedding space, while frames from the same video but taken from a different time have to be distant in the embedding space (24).
In a multi-view alignment task, the model must learn viewpoint-invariant representations of objects and other agents in their environment. The model endeavors to discern the dissimilarities between frames that occur later in the sequence by employing frames from the same video as negative pairs.

2.4.4 Time alignment

To perform self-supervised learning, two videos depicting the same type of action can be aligned in time. To learn the perfect alignment of two videos, a model has to be capable of concurrently disentangling phases of the activity in time and associating visually similar frames from the two distinct videos .
This self-supervised task produces representations useful for different downstream tasks that depend on fine-grained temporal features.

Dwibebi et al. (7) presented a self-supervised representation learning approach that performs temporal alignment across videos.
The proposed approach involves training a neural network by using temporal cycle-consistency (TCC),a loss function that is differentiable and enables the identification of temporal correspondences across numerous videos.
The objective is to acquire knowledge of an embedding space that optimizes the correspondence between pairs of video sequences belonging to the same action category. This objective can be achieved by maximizing the number of cycle-consistent frames between two sequences.
To check if a point $u_i \in U$ is cycle consistent, first its nearest neighbor, $v_j = \arg\min_{v \in V} \|u_i - v\|$, has to be found. Then the process is repeated to determine the nearest neighbor of vj in U, i.e. $u_k = \arg\min_{u \in U} \|v_j - u\|$. The point u_i is cycle-consistent if and only if i $=$ k,i.e. if the point u_i cycles back to itself.
Positive and negative examples of cycle-consistent points in an embedding space are presented in Figure 2.10.
Experimental results showed that the learned embeddings can be used to perform a few-shot classification of action phases, resulting in a substantial reduction in the need for supervised training.

The study conducted by Haresh et al.(25) introduced a self-supervised method for acquiring video representations through temporal video alignment. This strategy involved leveraging both frame-level and video-level information as a pretext task.
A unique approach that incorporated both temporal alignment loss and temporal

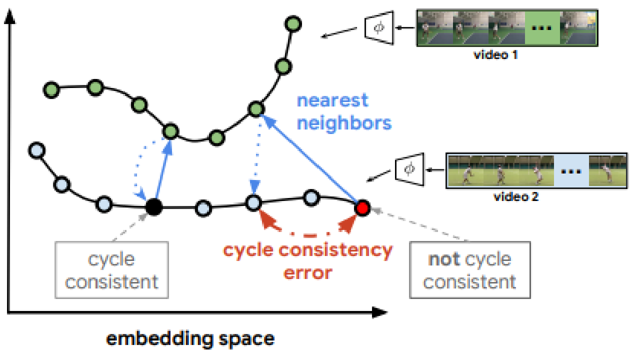


Figura 2.10: Cycle-consistent representation learning. Image taken from (7)

regularization terms is used as supervision signals to train an encoder network.
The primary objective of the temporal alignment loss is to minimize the cost associated with aligning videos in the embedding space with respect to time. Nevertheless, if the optimization process focuses exclusively on this term, it may result in unimportant solutions. Indeed, it could lead to a situation where all frames are assigned to a compact cluster inside the embedding space. In order to address this issue, a temporal regularization term was proposed. This term promotes mapping distinct frames to distinct points inside the embedding space.
The temporal alignment loss in this study was implemented using the traditional DTW discrepancy (26), while regularization was achieved through the utilization of the Inverse Difference Moment (IDM) (27).
While TCC (Temporal Consistency Constraints) aligns each frame individually, that approach focuses on aligning the video by utilizing both frame-level and video-level cues.

Liu et al. (13) employed an optimal transport matrix to align videos. Temporal priors are incorporated into the optimal transport matrix to prioritize the transporta- tion of one sequence to the elements located in close temporal positions of the other sequence. This approach ensures the preservation of the overall temporal structure and consisten- cy in the order of actions across sequences.

Chapter 3

Diffusion probabilistic models

3.1 Generative probabilistic models

Probabilistic models are designed to learn complex probability distributions.
By generating new samples, generative probabilistic models seek to gain an understanding of the target distribution. Specifically, the objective of generative probabilistic models is to discover a function that transforms data from a basic distribution into data from a complex target distribution.
Popular approaches inside the generative models family are:

- *Generative Adversarial Networks (GANs)*: these models learn the sampling procedure of complex distributions in an adversarial way.
  A generative network is trained to produce samples from a random input so that they are indistinguishable from the original samples of the target distribution.
  A discriminative network is trained to distinguish between original data and generated data.
  During the training process the generative network has to produce data that are always more conform to the target distribution to deceive the discriminative network.

- *Likelihood-based Models*: Likelihood-based models are designed to learn models that assign high probabilities to observed data samples. Examples include autoregressive models, normalizing flows, and Variational Autoencoders (VAEs).

- *Score-based Generative Models*: Score-based generative models are generative models that learn to generate data by estimating a score or gradient concerning the model's parameters.
  Unlike classic generative models like Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs), these models seek to directly approximate the data distribution without explicitly modeling the probability distribution.
  The central idea of score-based generative models is to compute the score (gradient) of the log-likelihood with respect to the model's parameters.
  Score-based models utilize gradient ascent to optimize model parameters. Given the actual data distribution, this process encourages the model to produce more probable data.

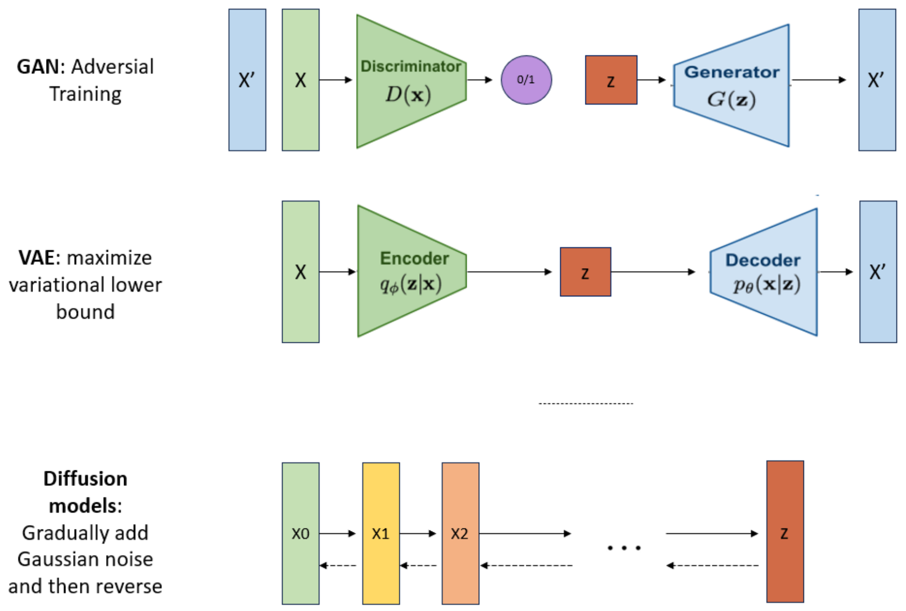

GAN: Adversial Training
X'
X
Discriminator $D(\mathbf{x})$
0/1
z
Generator $G(\mathbf{z})$
X'
VAE: maximize variational lower bound
X
Encoder $q_\phi(\mathbf{z}|\mathbf{x})$
z
Decoder $p_\theta(\mathbf{x}|\mathbf{z})$
X'
Diffusion models: Gradually add Gaussian noise and then reverse
X0
X1
X2
...
z

Figura 3.1: Quick overview of generative models

3.1.1 Variational Autoencoders (VAEs)

Variational autoencoders (VAEs) are one of the most used generative probabilistic models. They were introduced in 2013 by Kingma et al. (28). They posses an encoder-decoder design similar to conventional autoencoders.
The encoder function receives an input data point and transforms it into lower-dimensional latent space, a continuous, normally distributed space. Subsequently, the decoder maps points originating from the latent space back to the data space to reconstruct the original input.
The use of probabilistic modeling is the main innovation in VAEs. VAEs map data points to a probability distribution over the latent space instead of encoding them into a fixed point in the latent space. The encoder specifically outputs the mean and variance of a Gaussian distribution for each data point's location in the latent space. Therefore, the decoder must learn a function to map data from a Gaussian distribution into data from the original distribution.
The training objective for VAEs is to maximize the evidence lower bound (ELBO), which is a lower bound on the log-likelihood of the data.
The evidence lower bound can be computed as follows:

$$E_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{q_\phi(z \mid x)}\right] = E_{q_\phi(z|x)}\left[\log \frac{p_\theta(x \mid z)p(z)}{q_\phi(z \mid x)}\right]$$
(Chain Rule of Probability)

which can be further decomposed into:

$$E_{q_\phi(z|x)}\left[\log \frac{p_\theta(x \mid z)p(z)}{q_\phi(z \mid x)}\right] = E_{q_\phi(z|x)}\left[\log p_\theta(x \mid z)\right] - D_{KL}\left(q_\phi(z \mid x) \| p(z)\right)$$

where D_{KL} is the Kullback-Leibler Divergence.

The first term can be defined as a reconstruction loss that measures how well the decoder reconstructs the input.
The second term computes the Kullback-Leibler (KL) divergence between the distribution learned by the encoder and the prior distribution over the latent space. It ensures that the learned latent representations match a prior distribution (typically a standard Gaussian).
An intermediate bottlenecking distribution $q_\phi(z \mid x)$ is acquired within this framework and can be conceptualized as an "encoder." It generates a distribution over possible latent variables from inputs. Concurrently, a deterministic function $p_\theta(x \mid z)$ is learned, which functions as a "decoder" by transforming a specified latent vector z into an observation x.
To generate new data after training a VAE, one must first perform a direct sampling from the latent space $p(z)$ and subject it to decoder processing.
VAEs are particularly intriguing when the dimensionality of z is less than that of the input x, as this indicates the learning of meaningfully compact representations.
Further, acquiring a semantically significant latent space enables the modification of latent vectors before their transmission to the decoder, thereby facilitating a higher degree of accuracy in regulating the produced data.

3.1.2 Hierarchical Variational Autoencoders (HVAEs)

An extension of the conventional VAE, the Hierarchical Variational Autoencoder (HVAE) applies multiple hierarchies to latent variables.
Within this formulation, latent variables are not merely perceived as intermediary representations; rather, they are understood to originate from additional latent variables that are more abstract and higher-level.
Each latent variable in a general HVAE with T hierarchical levels may condition on every preceding latent.
HVAEs can produce data at various levels of abstraction and time scales, ranging from the coarsest level down to the finest level. This structured and hierarchical generation process enables the production of complex sequences or data.
A special family of HVAEs are Markovian HVAEs (MHVAEs). They use a Markov Chain as generative process. Therefore each latent z_t is generated only from the previous latent z_{t+1}.

3.2 Basic concept of Diffusion Probabilistic models

Diffusion probabilistic models were introduced in 2015 by Sohl-Dickstein et al. (29). Motivated by the principles of non-equilibrium statistical physics, the fundamental concept entails employing an iterative forward diffusion process to methodically and gradually destroy structure from a given data distribution. Then, the model has to learn a process of reverse diffusion that restores data structure, resulting in a generative model of the data that is extraordinarily adaptable and manageable.
The diffusion Model can be best understood as a special case of a Markovian Hierarchical Variational Autoencoder. However, it comes with three distinct characteristics:

1. Latent spaces have the same dimension as the original space.

2. During the training the structure of the latent encoder at each timestep has not to be learned. Instead, it is pre-defined as a linear Gaussian model. This implies that the distribution follows a Gaussian distribution with its center located at the output of the preceding time step.

3. The Gaussian parameters of the latent encoders change over time. This modification guarantees the latent distribution at the ultimate timestep T conforms to a standard Gaussian distribution.

The main structure of the entire process can be shown in the figure 3.2.

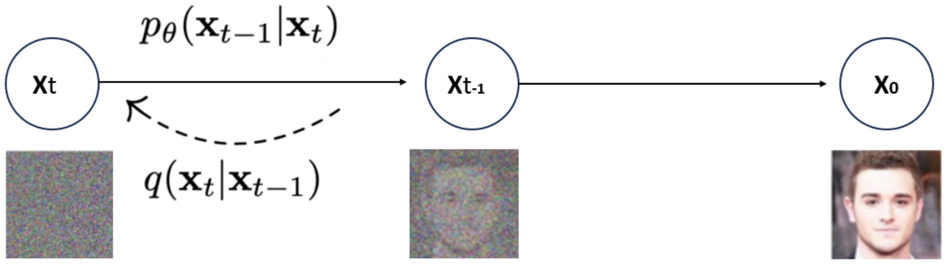

$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
Xt
Xt-1
X0
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$

Figura 3.2: Diffusion Probabilistic model

In the forward diffusion process, data from the original distribution are gradually brought to a simple Gaussian distribution. In the reverse process, Gaussian noise is taken as an input, and data from the distribution of interest are generated. So, in the reverse process, the model learns a function that transforms data from a basic distribution into data from a complex target distribution.
The introduction of several intermediate steps between the target and simple distribution makes the reverse process relatively easy to learn. It would be very complicated to learn the relative process without relying on the intermediate steps since, looking at some given noise, not much information can be found. However, by examining intermediate stages, certain indicators can be employed to direct the model toward reversing the process.

3.2.1 Forward Diffusion Process

In the forward diffusion process, some noise is slowly and iteratively added to the original data such that the original distribution is converted to a basic distribution.
In figure 3.1 the term $q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)$, also known as the forward diffusion kernel (FDK), denotes the "transition function" applied at each step in the forward diffusion process. During the diffusion process, the data distribution is gradually converted into a well-behaved (analytically tractable) distribution $\pi(y)$ by repeated application of a Markov diffusion kernel $T_\pi\left(\mathbf{y} \mid \mathbf{y}'; \beta\right)$ for $\pi(y)$:

$$\pi(\mathbf{y}) = \int d\mathbf{y}' T_\pi\left(\mathbf{y} \mid \mathbf{y}'; \beta\right) \pi\left(\mathbf{y}'\right)$$

where $\beta \in (0,1)$ is the diffusion rate and controls the step size.
Given $q\left(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-1)}\right) = T_\pi\left(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-1)}; \beta_t\right)$, The forward trajectory from the original data by computing t steps is computed as following:

$$q\left(\mathbf{x}^{(0\cdots T)}\right) = q\left(\mathbf{x}^{(0)}\right) \prod_{t=1}^{T} q\left(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-1)}\right)$$

The following Markov diffusion kernel is employed to transform the data distribution into a Gaussian distribution progressively:

$$\mathcal{N}\left(x^{(t)}; x^{(t-1)}\sqrt{(1-\beta_t)}, I\beta_t\right)$$

Therefore, the data at time step t can be computed as follows:

$$x_t = \sqrt{1-\beta_t} x_{t-1} + \sqrt{\beta_t}\epsilon_t$$

If the data at time step t is computed in this way, t-1 steps of the Markov chain must be executed whenever a latent sample x at timestep t is required.
To obtain data more efficiently, Sohl-Dickstein et al.(29) introduced a **reparameterization trick** to go from timestep 0 to timestep t directly.
And then, by substituting β_t with α_t and using the addition property of Gaussian distribution, the forward diffusion process can be rewritten as follows:

$$q\left(x_t \mid x_0\right) := \mathcal{N}\left(x_t; \sqrt{\bar{\alpha}_t} x_0, \left(1-\bar{\alpha}_t\right) I\right)$$

Therefore, the data at time step t can be computed as follows:

$$x_t := \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon_t$$

where $\alpha_t = (1-\beta_t)$ and $\bar{\alpha}_t = \prod_{t=1}^{t} \alpha_t = \prod_{t=1}^{t} (1-\beta_t)$

3.2.2 Reverse Diffusion Process

The reverse diffusion process aims to remove noise introduced in the forward diffusion process. Slowly and iteratively some transformations are used to try to reconstruct the original input from the output of the forward process.
If we can reverse the above process and sample from $p_\theta\left(x_{t-1} \mid x_t\right)$, we will be able to recreate the true sample from a Gaussian noise input,$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
$p_\theta\left(x_{t-1} \mid x_t\right)$, known as the reverse diffusion kernel (RDK), is the transition function applied at each step in the reverse diffusion process. As demonstrated by Feller et al.(30) for Gaussian diffusion, if the step size B is small, the diffusion process's reversal has the same functional form as the forward process. Therefore:

$$p\left(x_T\right) := \mathcal{N}\left(x_t; 0, I\right)$$

and the Markov diffusion kernel is the following:

$$p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right) := \mathcal{N}\left(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right), \boldsymbol{\Sigma}_\theta\left(\mathbf{x}_t, t\right)\right)$$

The reverse trajectory from the data of the forward process at time step t to the original data is thus:

$$p_\theta\left(\mathbf{x}_{0:T}\right) := p\left(\mathbf{x}_T\right) \prod_{t=1}^{T} p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)$$

Multi-layer perceptrons are usually used to learn the mean and covariance of the reverse Markov transition.

3.2.3 Training

If a Gaussian diffusion kernel is used, only the diffusion kernel's mean and covariance must be estimated during training.
Training is performed by optimizing the variational bound on negative log-likelihood. Different transformations are used to obtain the objective function(8):

$$\mathbb{E}\left[-\log p_\theta\left(\mathbf{x}_0\right)\right] \leq \mathbb{E}_q\left[-\log \frac{p_\theta\left(\mathbf{x}_{0:T}\right)}{q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right)}\right]$$

$$\begin{aligned}
L &= \mathbb{E}_q\left[-\log \frac{p_\theta\left(\mathbf{x}_{0:T}\right)}{q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right)}\right] \\
&= \mathbb{E}_q\left[-\log p\left(\mathbf{x}_T\right)-\sum_{t \geq 1} \log \frac{p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)}{q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)}\right] \\
&= \mathbb{E}_q\left[-\log p\left(\mathbf{x}_T\right)-\sum_{t>1} \log \frac{p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)}{q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)}-\log \frac{p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}{q\left(\mathbf{x}_1 \mid \mathbf{x}_0\right)}\right] \\
&= \mathbb{E}_q\left[-\log p\left(\mathbf{x}_T\right)-\sum_{t>1} \log \frac{p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)}{q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right)} \cdot \frac{q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_0\right)}{q\left(\mathbf{x}_t \mid \mathbf{x}_0\right)}-\log \frac{p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}{q\left(\mathbf{x}_1 \mid \mathbf{x}_0\right)}\right] \\
&= \mathbb{E}_q\left[-\log \frac{p\left(\mathbf{x}_T\right)}{q\left(\mathbf{x}_T \mid \mathbf{x}_0\right)}-\sum_{t>1} \log \frac{p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)}{q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right)}-\log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)\right] \\
=& \mathbb{E}_q\left[D_{\mathrm{KL}}\left(q\left(\mathbf{x}_T \mid \mathbf{x}_0\right) \| p\left(\mathbf{x}_T\right)\right)+\sum_{t>1} D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)-\log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)\right]
\end{aligned}$$

Therefore the new objective function is the following one:

$$\mathbb{E}_q[\underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_T \mid \mathbf{x}_0\right) \| p\left(\mathbf{x}_T\right)\right)}_{L_T}+\sum_{t>1} \underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)}_{L_{t-1}} \underbrace{-\log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}_{L_0}]$$

It is made up of three terms.
The first term refers to the "KL divergence" between the distribution of the final latent variable in the forward process and the first latent variable in the reverse process. It shows how close $\mathbf{x}T$ is to the standard Gaussian. The entire term has no trainable parameters, so it's ignored during training.
The second term refers to the difference between the desired denoising steps and the approximated ones.
The third term can be seen as a reconstruction term, similar to the one in the ELBO of a variational autoencoder. In Ho et al. (8), this term is learned using a separate decoder derived from the Gaussian $\mathcal{N}\left(\mathbf{x}_0; \boldsymbol{\mu}_\theta\left(\mathbf{x}_1, 1\right), \sigma_1^2 \mathbf{I}\right)$:

$$p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)=\prod_{i=1}^D \int_{\delta_-\left(x_0^i\right)}^{\delta_+\left(x_0^i\right)} \mathcal{N}\left(x; \mu_\theta^i\left(\mathbf{x}_1, 1\right), \sigma_1^2\right) dx$$

$$\delta_+(x)=\begin{cases}\infty & \text{if } x=1 \\ x+\frac{1}{255} & \text{if } x<1\end{cases} \quad \delta_-(x)=\begin{cases}-\infty & \text{if } x=-1 \\ x-\frac{1}{255} & \text{if } x>-1\end{cases}$$

where D is the data dimensionality and the i superscript indicates extraction of one coordinate.
Regarding the second term of the objective function, even though $q\left(\mathbf{x}_{t-1}\ \mathbf{x}_t\right)$ is intractable as calculating it involves the whole data distribution, Sohl-Dickstein et al.(29) illustrated that by additionally conditioning on x0 makes it tractable:

$$q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right)=\mathcal{N}\left(\mathbf{x}_{t-1} ; \tilde{\boldsymbol{\mu}}\left(\mathbf{x}_t, \mathbf{x}_0\right), \tilde{\beta}_t \mathbf{I}\right)$$
$$\tilde{\beta}_t=\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t$$
$$\tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right)=\frac{\sqrt{\bar{\alpha}_{t-1}} \beta_t}{1-\bar{\alpha}_t} \mathbf{x}_0+\frac{\sqrt{\alpha_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t} \mathbf{x}_t$$

As previously seen in the **reparameterization trick**, x0 can be reformulated as:

$$\mathbf{x}_0=\frac{1}{\sqrt{\bar{\alpha}_t}}\left(\mathbf{x}_t-\sqrt{1-\bar{\alpha}_t} \boldsymbol{\epsilon}_{\boldsymbol{t}}\right)\Big)$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
Therefore a neural network can be used to compute $p_\theta\left(x_{t-1} \mid x_t\right)$. Specifically it can be used to approximate $\boldsymbol{\epsilon}$ and consequently the mean:

$$\tilde{\boldsymbol{\mu}}_\theta\left(\mathbf{x}_t, t\right)=\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t-\frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta\left(\mathbf{x}_t, t\right)\right)$$

Thus, the loss function can be parameterized to minimize the difference from the means as follows:

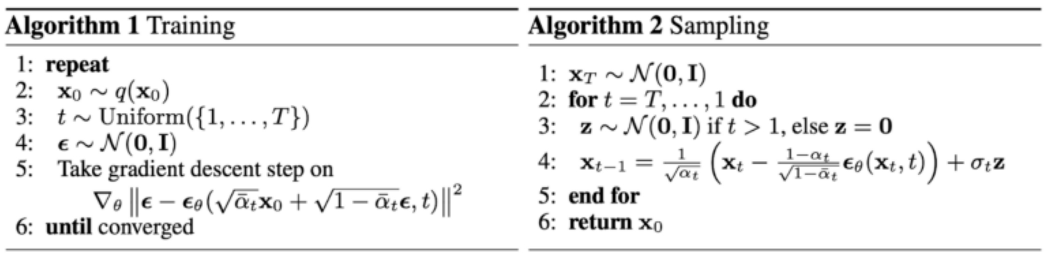
$$\begin{aligned}
L_t & =\mathbb{E}_{\mathbf{x}_0, \epsilon}\left[\frac{1}{2\left\|\boldsymbol{\Sigma}_\theta\left(\mathbf{x}_t, t\right)\right\|_2^2}\left\|\tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right)-\boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right)\right\|^2\right] \\
& =\mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}}\left[\frac{1}{2\left\|\boldsymbol{\Sigma}_\theta\right\|_2^2}\left\|\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t-\frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_t\right)-\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t-\frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta\left(\mathbf{x}_t, t\right)\right)\right\|^2\right] \\
& =\mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}}\left[\frac{\left(1-\alpha_t\right)^2}{2 \alpha_t\left(1-\bar{\alpha}_t\right)\left\|\boldsymbol{\Sigma}_\theta\right\|_2^2}\left\|\boldsymbol{\epsilon}_t-\boldsymbol{\epsilon}_\theta\left(\mathbf{x}_t, t\right)\right\|^2\right] \\
& =\mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}}\left[\frac{\left(1-\alpha_t\right)^2}{2 \alpha_t\left(1-\bar{\alpha}_t\right)\left\|\boldsymbol{\Sigma}_\theta\right\|_2^2}\left\|\boldsymbol{\epsilon}_t-\boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t} \mathbf{x}_0+\sqrt{1-\bar{\alpha}_t} \boldsymbol{\epsilon}_t, t\right)\right\|^2\right]
\end{aligned}$$

Empirically, Ho et al.(8) found that training the diffusion model works better with a simplified objective that ignores the weighting term:

$$\begin{aligned}
L_t^{\text{simple}} & =\mathbb{E}_{t \sim[1, T], \mathbf{x}_0, \epsilon_t}\left[\left\|\boldsymbol{\epsilon}_t-\boldsymbol{\epsilon}_\theta\left(\mathbf{x}_t, t\right)\right\|^2\right] \\
& =\mathbb{E}_{t \sim[1, T], \mathbf{x}_0, \epsilon_t}\left[\left\|\boldsymbol{\epsilon}_t-\boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t} \mathbf{x}_0+\sqrt{1-\bar{\alpha}_t} \boldsymbol{\epsilon}_t, t\right)\right\|^2\right]
\end{aligned}$$

3.2.4 Denoising Diffusion Implicit Models

A considerable drawback of DDPMs is the extensive number of iterations necessary to generate high-quality samples. This is because the generative process of DDPMs approximates the reverse of the forward diffusion process, which can have thousands of



Algorithm 1 Training

```
1: repeat
2:   $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3:   $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
4:   $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5:   Take gradient descent step on
         $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$
6: until converged
```

Algorithm 2 Sampling

```
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: for $t = T, \ldots, 1$ do
3:   $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4:   $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: end for
6: return $\mathbf{x}_0$
```

Figura 3.3: Training and sampling algorithms for a DDPM, as shown in Ho et al., 2020 (8)

steps.
To overcome the limits of DDPMs, DDIMs (Denoising Diffusion Implicit Models) were introduced. DDIMs are implicit probabilistic models closely associated with DDPMs since they share the same objective function during training. DDIM's main benefit is that it enables significantly faster sampling while keeping the same training objective. Instead of making an approximation of the transition from x_t to x_{t-1}, DDIM employs an approximation method that combines x_0 with a noise variable in order to generate x_t.
By utilizing the identical training objective and model as DDPM, DDIM can achieve a much faster image reconstruction process by limiting the number of steps taken in the Markov chain.
The DDIM samples exhibit a quality known as "consistency," indicating that when the same initial latent variable is used to generate many samples using Markov chains of different lengths, these samples will have comparable high-level characteristics.
The consistent nature of Deep Deterministic Implicit Models (DDIMs) allows for manipulating the initial latent variable to execute semantically meaningful image interpolation. On the other hand, DDPMs have suboptimal interpolation capabilities.

Advantages and challenges: Although diffusion models have demonstrated considerable potential, they are not without their unique challenges. Different advantages can be identified:

- **Flexibility and generative power**: Diffusion models successfully produce high-quality samples by modeling various complex data distributions.
- **Interpretability**: they offer an intelligible mathematical framework for comprehending data distributions that can be applied to domains other than images.

which come at some costs nonetheless:

- **Computational complexity** : The computational cost of the reverse diffusion process can be substantial, and the training phase typically requires a significant amount of time.
- **Training stability** : Diffusion models exhibit high sensitivity to hyperparameters and training conditions, hence making the training process more challenging to manage.
- **Scalability**: Executing large-scale applications may require substantial computer resources.

In deep learning, diffusion models offer a robust and adaptable framework for modeling complex data distributions. Through comprehension of the fundamental diffusion process, these models possess the capability to generate novel samples and forecast data trajectories.

3.3 Relevant works on diffusion

Diffusion models belong to a class of deep generative models that have gained significant attention in computer vision. These models have demonstrated remarkable generative skills, exhibiting a wide range of high-level details and diversity in the instances they generate.
Thus far, diffusion models have been utilized in a diverse range of generative modeling tasks, including but not limited to image generation (8; 31), image super-resolution (32), image inpainting (9) and text-to-image synthesis (33) among various others.
Furthermore, it has been observed that the latent representation acquired using diffusion models has proven to be valuable in discriminative tasks, such as image segmentation (10) and classification (34).

In the context of image generation, diffusion models can generate images in two ways: unconditionally, without any supplementary information (8), or conditionally (31), by including specific information. The condition typically relies on various source signals, with class labels often utilized.
The concept of conditional **image generation** was initially proposed by Dhariwal et al. (31). They introduced augmented diffusion models guided by classifiers. This approach employs the gradients of a classifier to direct the sampling diffusion. By following the guidance of this classifier, images associated with specific class labels can be generated.

The objective of **image super-resolution** is to convert low-resolution inputs into high-resolution images. Numerous approaches employ diffusion models to accomplish these objectives. For instance, DDPM is utilized by Saharia et al. (32) to facilitate the generation of conditional images.
Super-resolution is achieved by Saharia et al. (32) via an iterative, stochastic denoising procedure. In this case, the diffusion process is explicitly applied to input images, resulting in more extensive evaluation steps.
To allow for the training of diffusion models with limited computational resources, the Latent Diffusion Model (35) has shifted the diffusion process to the latent space using pre-trained autoencoders. Performance obtained in super-resolution task is comparable to that obtained by Saharia et al.(32).

The process of **image inpainting** involves reconstructing parts within an image that are either absent or damaged. In the context of inpainting activities, the generation of new image content relies only on the information provided by the non-masked pixels.
RePaint is a diffusion model that performs image inpainting. It was introduced in 2022 by Lugmayr et al.(9). This study's conditioning procedure involves sampling from the given pixels during the reverse diffusion iterations. An unconditional diffusion model is employed, with modifications made to its reverse process.
The picture at step t - 1 is generated by sampling the known region from the masked image. In contrast, the unknown region is obtained by applying denoising techniques

to the image obtained at step t (see Figure 3.4).

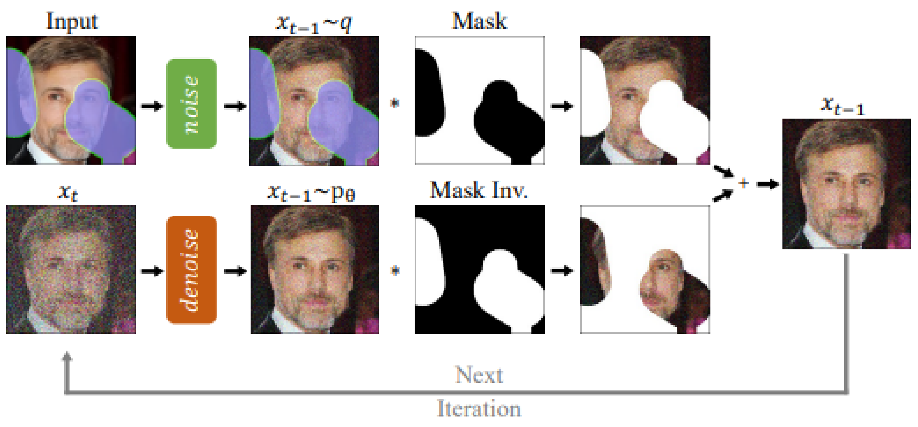


Figura 3.4: Method proposed by Lugmayr et al.(9). Image taken from (9)

By implementing this methodology, the researchers ascertain that the unidentified area exhibits the appropriate configuration, albeit displaying semantic inaccuracies. Moreover, the problem is addressed by iteratively executing the suggested procedure multiple times.
During each iteration, the previous image from step t is substituted with a fresh sample derived from the denoised version produced at step t − 1.

Text-to-image synthesis refers to generating images from textual descriptions or captions. The task of text-to-image synthesis was initially proposed by Nichol et al. (33).
Conditions are applied to natural language descriptions using a text encoder. Following this, images conditioned on the text embeddings returned by the encoder are generated using a diffusion model.

Baranchuk et al. (10) illustrate the application of diffusion models to **semantic segmentation**.
In this work, latent representation acquired using diffusion models is used to perform image segmentation as described in Figure 3.5.
A data representation is obtained by concatenating the feature maps obtained at various scales from the decoder of the U-Net, which is utilized in the denoising process, and upsampling the feature maps to achieve identical dimensions. Then, an ensemble of multi-layer perceptrons can be appended to classify each pixel further.
Mukhopadhyay et al.(34) showed that the intermediate feature maps of the U-Net architecture can also be used to perform an image classification task.

Recently, the diffusion model was used to perform **positional reasoning**, which is the process of arranging unsorted parts within a set into a consistent structure.
Giuliari et al.(11) proposed an innovative forward and reverse diffusion process formulation to correctly sort a shuffled input. The positions of elements in a set are mapped to random places in a continuous space using a forward procedure. Using

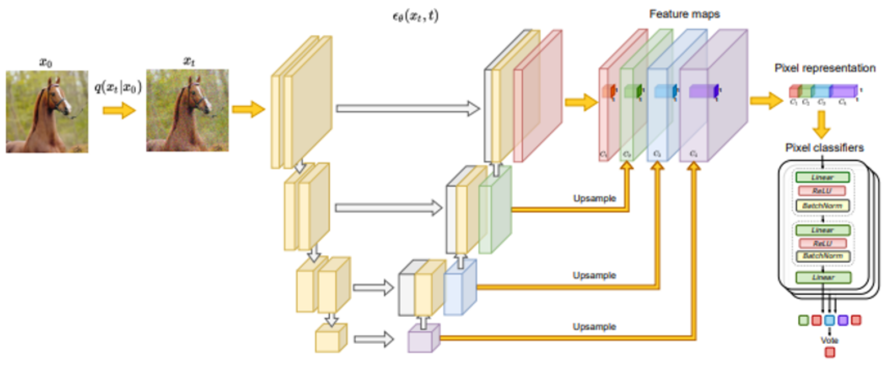


Figura 3.5: Method proposed by Baranchuk et al.(10)

an Attention-based Graph Neural Network, the Positional Diffusion model learns to reverse the noise-generating process and restore the original positions.
Through a comprehensive examination of multiple benchmark datasets, the authors demonstrated that their approach surpassed existing deep learning frameworks at solving puzzles while performing on par with state-of-the-art techniques about sentence ordering and visual storytelling.
The authors execute positional reasoning as a procedure for recovering shuffled, unstructured data in an Euclidean space R^n, where n=1 for one-dimensional problems such as sentence ordering and n = 2 for two-dimensional tasks such as solving puzzles. Giuliari et al.(11) have used a task-specific backbone to obtain an informative representation of the input data.
Given an unordered set of K elements containing task-specific features denoted as $\mathbf{H} = \left\{\mathbf{h}^1, \ldots, \mathbf{h}^K\right\}, \mathbf{h}^i \in \mathbb{R}^d$ where d represents the dimension of the features, and ground truth positions $\mathbf{X} = \left\{\mathbf{x}^1, \ldots, \mathbf{x}^K\right\}, \mathbf{x}^i \in \mathbb{R}^n$, a network is used to generate an approximation of the positions of each element $\hat{\mathbf{X}} = \left\{\hat{\mathbf{x}}^1, \ldots, \hat{\mathbf{x}}^K\right\}, \hat{\mathbf{x}}^i$.
Each data point is represented as a node in a fully connected graph, thereby enabling the influence of each node on the others.
Graph representation has the advantageous characteristic of being able to accommodate a variable quantity of input data in a manner that is invariant to permutations.
In order to address the reverse problem, a neural network is trained. The noise t, which is utilized in the calculation of $\mathbf{X}_{t-1}$, is determined based on the given noisy positions $\mathbf{X}_t$, features H, and time step t.
The network mentioned above utilizes element features, denoted as $\mathbf{h}^i$, which can be derived from a pre-trained backbone specific to the task to solve. A graph structure, denoted as G, was constructed, whereby each node corresponds to an input point and $\left[\mathbf{x}_t^i; \mathbf{h}^i\right]^\top$ is assigned as node features.
That graph is the input for an Attention-based Graph Neural Network (GNN) backbone. This backbone consists of a series of four layers of Graph Transformers(36) .
Graph Transformers employ an attention mechanism in conjunction with a graph structure to regulate the quantity of information extracted from adjacent nodes. The ability to focus is indeed what enables the neural network to acquire the capability of doing positional reasoning.
Data shuffling was interpreted as the noise injection of DPMs' forward process, and

the reverse process of a DPM is exploited to retrieve the final position of each element.
The authors assess Positional Diffusion across various tasks, including the resolution

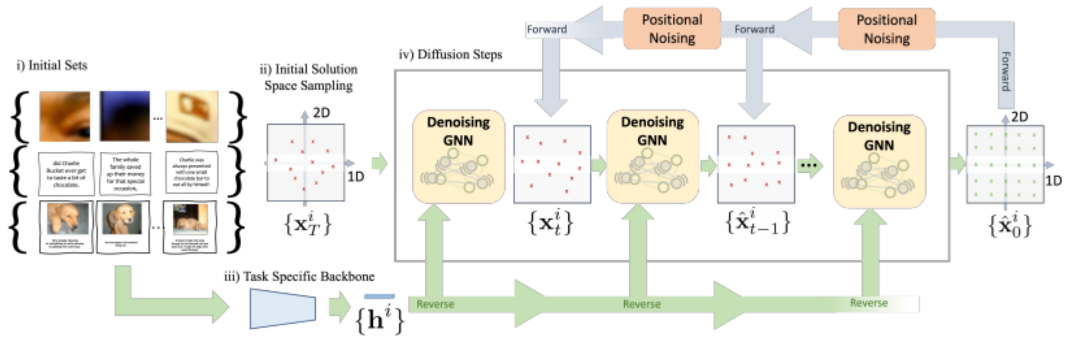


Figura 3.6: The model's base architecture proposed by the Positional Diffusion authors. Image taken from(11)

of puzzles.
The researchers assess the performance of Positional Diffusion in relation to deep learning and optimization-based approaches using two datasets: PuzzleCelebA and PuzzleWikiArts. The input data consists of pixel values for each patch, resized to 32x32. An EfficientNet was used as the task-specific backbone to extract the visual features of the patch.
T = 300 iterations are used to train the diffusion model, while r = 10 is sampled for the inference ratio. The authors simultaneously train a single model for every puzzle size.
In the inference process, the patches are organized by assigning a cell in a grid to each estimated patch position. The distance between each patch position and the cells' centers is measured, and each patch is assigned to its closest cell.
In addition, the authors evaluated the model on visual storytelling and sentence ordering tasks, where the model utilized a text-to-marker framework to generate an ordered output sequence of sentences from a randomized input sequence.
A significant advantage of Positional Diffusion is its generic nature. The approach is versatile and can be implemented in numerous tasks that require positional reasoning, regardless of the dimensionality of data modality (e.g., image, text) or the positional problem (e.g., 1D for text, 2D for images).
In essence, the Positional Diffusion technique provides a comprehensive, adaptable, and efficient solution for addressing diverse ordering challenges, showcasing its resilience across various modalities and datasets.

Chapter 4

Introduction to graphs

The proposed approach uses the diffusion process within the graph domain. Indeed, the position of each clip and the corresponding feature representation is encoded as a node in a graph. The graph thus constructed serves as input for the diffusion process. Therefore, an introduction to graphs and graph neural networks is necessary before the proposed approach can be explored in detail.

Types of Graphs. A graph is a mathematical structure used to represent relationships between objects. It is composed of nodes and edges.
Generally, an entity is denoted by a single node, while a single edge represents a relationship or connection between two entities.
Graphs can be classified according to the characteristics of the connections or relationships that exist between their nodes.

- **Directed vs. Undirected Graphs**: An undirected graph is a type of graph where the edges do not have any orientation. This means that the edge (x, y) is considered the same as the edge (y, x).
  On the other hand, a graph that has a distinct orientation of its edges is commonly known as a directed graph or a digraph.
- **Cyclic vs. Acyclic Graphs**: A graph is considered cyclic if it has at least one cycle, defined as a path consisting of edges and vertices where a node can be reached from itself. In the absence of cycles, the graph is purely acyclic.

4.1 Graph Neural networks (GNNs)

Graph neural networks (GNNs) are deep learning-based methods that operate on the graph domain.
Conventional machine learning algorithms handle graph-structured data through a preprocessing stage that converts the graph-structured information to a more straightforward format, such as vectors of real numbers(37).
In other words, the preprocessing phase "squashes" the graph-structured data into a vector of reals before applying a list-based data processing technique to the preprocessed data.
Nevertheless, crucial data, such as the topological dependence of information on individual nodes, might be compromised throughout the preprocessing phase. Consequently,

the outcome might be influenced by the details of the preprocessing algorithm.
Graph neural networks were first introduced by Scarselli et al.(38).
They are based on the intuitive notion that nodes in a graph symbolize entities or concepts while edges delineate their connections.
The definition of each concept is inherently determined by its characteristics and the concepts to which it is related.
Therefore, it is possible to assign a state to each node depending on the information in its surrounding neighborhood.
Consequently, the goal of graph neural networks is to learn a function f that computes the state of each single node as follows:

$$\mathbf{h}_v = f\left(\mathbf{x}_v, \mathbf{x}_{co[v]}, \mathbf{h}_{ne[v]}, \mathbf{x}_{ne[v]}\right)$$

where $\mathbf{x}_{co[v]}$ denotes the features of the edges connecting with v, $\mathbf{h}_{ne[v]}$ denotes the embedding of the neighboring nodes of v, and $\mathbf{x}_{ne[v]}$ denotes the features of the neighboring nodes of v.
This particular transformation is commonly known as message passing.
Message passing for each node in the graph involves three stages:

1. *Messaging*:the embeddings of the adjacent nodes are collected;

2. *Aggregation*: the aggregation of all messages is achieved by a permutation invariant mechanism function (e.g., sum or mean);

3. *Update*: every pooled message is run through an update function, often a learned neural network.

The most basic form of message passing GNN layer exclusively employs this collection of operations once.
This emulates a traditional convolution. Indeed both message passing and convolutions involve the collection and processing of data from the neighbors of an element, which is then used to alter the value of that element (where "element" can refer to a pixel in an image or a node in a graph).
As opposed to an image where a constant number of neighbors surrounds each pixel, the number of neighboring nodes in a graph can vary.
By accumulating message-passing layers, a node can ultimately incorporate information from throughout the entire graph; for instance, after three levels, a node possesses knowledge of the nodes that are three steps distant from it.
To calculate the output of the GNN, the state $\mathbf{h}_v$ and the feature $\mathbf{x}_v$ are passed to an output function g:

$$\mathbf{o}_v = g\left(\mathbf{h}_v, \mathbf{x}_v\right)$$

Node representation and edge prediction are two of the most prevalent applications of GNNs.
Edge prediction is the process of forecasting whether or not an edge connects two elements in a graph. This is applicable to numerous domains, including bioinformatics (e.g., predicting protein interactions), social network analysis (e.g., determining the likelihood that two individuals will share a particular relationship), and recommendation systems (e.g., predicting whether a user will like a particular item).
Node prediction, conversely, pertains to the prediction of node-specific attributes (e.g., determining a user's human or machine status in a social network or the subject matter of a paper in a citation network).

Node Prediction . The task of node-level prediction involves the prediction of properties or labels assigned to individual nodes within a network, taking into consideration their position within the graph structure and their connections with other nodes.
The task of node prediction within the framework of Graph Neural Networks (GNNs) often involves training a function that maps a node's representation to a set of labels or continuous values.
The Graph Neural Network (GNN) is trained to identify a representation for each node that effectively captures its essential attributes and role within the graph.
Once the process of acquiring node representations has been completed, it becomes possible to do node prediction by applying the prediction function to the respective node.
The choice of the prediction function may vary depending on whether the task involves classification or regression.
The selection of an appropriate prediction function in constructing a Graph Neural Network (GNN) is a critical design decision that can be influenced by the specific task to solve and the structural properties of the graph.

Edge Prediction. The task of edge prediction entails the prediction of properties or characteristics associated with each individual edge that connects a pair of nodes.
To accomplish this, the utilization of message passing is once again required.
Exploiting surrounding edge information can be achieved similarly to the previous incorporation of neighboring node information.
This involves pooling the edge information, applying an update function to transform it, and storing it.
It is not immediately obvious how to merge node and edge information because they are not always the same size or shape when recorded in a graph.
For example, linear mapping can be learned from the space of edges to the space of nodes and vice versa. Alternatively, one might combine them before the update function.
Regardless of the design decision, there are always two fundamental steps involved in an edge update:

- edge-to-node update, where each node receives information from neighboring edges;

- node-to-edge update, where each edge receives information from the two nodes that are connecting

4.2 Graph attention networks (GANs)

Graph attention networks were first introduced by Velickovic et al.(12).
The key idea behind GATs is to incorporate attention mechanisms into aggregating information from neighboring nodes in a graph. This allows nodes to weigh the importance of their neighbors differently when updating their representations.
The attention mechanism lets the network focus on relevant information, making it a powerful tool for handling graphs with varying connectivity patterns.
GAT employs an attention mechanism to compute attention scores for each node's neighbors.
These scores determine the importance of information from neighboring nodes.

The attention score of node j with respect to node i is computed as follows:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T\left[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j\right]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T\left[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k\right]\right)\right)}$$

where the attention mechanism a is a single-layer feedforward neural network, parametrized by a weight vector $\vec{\mathbf{a}}$, T represents transposition and $\|$ is the concatenation operation. A non-linear function σ is used to compute the final output features for each node as follows:

$$\vec{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\mathbf{W}\vec{h}_j\right)$$

Shi et al (36) proposed a different attention mechanism. This work uses the same

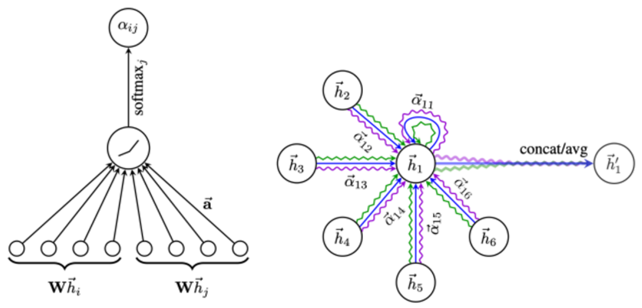


Figura 4.1: The attention mechanism $a\left(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j\right)$ employed by GAT (12) is shown on the left, parametrized by a weight vector $\vec{\mathbf{a}}$ and followed by a LeakyRelu activation function.
An example of multi-headed attention is presented on the right, wherein three-headed attention is applied by node one toward its neighborhood.
Different colored arrows are used to represent distinct attention heads.
To obtain $\vec{h}'_1$, the aggregated features from each head are concatenated or averaged.

attention mechanism introduced with the transformers (39).
Specifically, given node features $H^{(l)} = \left\{h_1^{(l)}, h_2^{(l)}, \ldots, h_n^{(l)}\right\}$, a multi-head attention is calculated for each edge from j to i as follows:

$$\begin{aligned}
q_{c,i}^{(l)} &= W_{c,q}^{(l)}h_i^{(l)} + b_{c,q}^{(l)} \\
k_{c,j}^{(l)} &= W_{c,k}^{(l)}h_j^{(l)} + b_{c,k}^{(l)} \\
e_{c,ij} &= W_{c,e}e_{ij} + b_{c,e} \\
\alpha_{c,ij}^{(l)} &= \frac{\left\langle q_{c,i}^{(l)}, k_{c,j}^{(l)} + e_{c,ij}\right\rangle}{\sum_{u\in\mathcal{N}(i)}\left\langle q_{c,i}^{(l)}, k_{c,u}^{(l)} + e_{c,iu}\right\rangle}
\end{aligned}$$

where $\langle q, k\rangle = \exp\left(\frac{q^T k}{\sqrt{d}}\right)$ is the exponential scale dot-product function and d is the hidden size of each head. For the c-th head attention, the source feature $h_i^{(l)}$ and

distant feature $h_j^{(l)}$ are firstly transform into query vector $q_{c,i}^{(l)} \in \mathbb{R}^d$ and key vector $k_{c,j}^{(l)} \in \mathbb{R}^d$ respectively using different trainable parameters $W_{c,q}^{(l)}, W_{c,k}^{(l)}, b_{c,q}^{(l)}, b_{c,k}^{(l)}$.
The edge features (eij) will be encoded and incorporated as supplementary information into the key vector for every layer.
Once the graph multi-head attention has been obtained, a message aggregation is performed from the distant j to the source i as follows:

$$\begin{aligned} v_{c,j}^{(l)} &= W_{c,v}^{(l)} h_j^{(l)} + b_{c,v}^{(l)} \\ \hat{h}_i^{(l+1)} &= \|_{c=1}^{C} \left[\sum_{j \in \mathcal{N}(i)} \alpha_{c,ij}^{(l)} \left(v_{c,j}^{(l)} + e_{c,ij} \right) \right] \end{aligned}$$

where || is the concatenation operation for C heads attention.
Furthermore, the model employs a gated residual connection between layers to mitigate the risk of over-smoothing:

$$\begin{aligned} r_i^{(l)} &= W_r^{(l)} h_i^{(l)} + b_r^{(l)} \\ \beta_i^{(l)} &= \text{sigmoid} \left(W_g^{(l)} \left[\hat{h}_i^{(l+1)}; r_i^{(l)}; \hat{h}_i^{(l+1)} - r_i^{(l)} \right] \right) \\ h_i^{(l+1)} &= \text{ReLU} \left(\text{LayerNorm} \left(\left(1 - \beta_i^{(l)} \right) \hat{h}_i^{(l+1)} + \beta_i^{(l)} r_i^{(l)} \right) \right) \end{aligned}$$

Chapter 5

Proposed approach

5.1 Overview

As mentioned earlier in the introduction, this work aims to use a self-supervised approach to learn a data representation useful for various downstream tasks.
A reordering task is used as a self-supervised task. Each video is divided into clips and these clips are then shuffled and reordered by the model.
A forward diffusion process is used to inject some noise in the correct position of clips. Then an Attention-base Graph Neural Network is used to learn the reverse diffusion process and restore the original position of each clip.
The usefulness of the learned data representation was investigated in an action classification task and a phase classification task

5.2 Literature review

In the proposed approach the same self-supervised task of the work of Xu et al. (6) is used. In (6) each video is divided into several clips. The clips are then shuffled and three-dimensional convolutional neural networks (3D CNNs) are employed to extract features from individual video clips. Then an order prediction task is performed as a classification task on the extracted features.
Whereas in the work of Xu et al.(6) the order prediction is performed as a classification task, in the proposed method, using a diffusion model, the task is performed as a regression task. Using a regression rather than a classification task is relevant because, in a classification task, the complexity of the task increases considerably as the number of elements to be reordered increases. Having n number of frames, the possible classes are n!. Therefore, using a regression task allows even a large number of elements to be reordered more efficiently.
Furthermore, in the case of a classification task, every error is evaluated in the same way. Even if the clips were partially sorted or positioned in the reverse order, this would be evaluated in the same way as if each clip were incorrectly positioned. On the other hand, the use of a regression task allows the use of other evaluation metrics that also consider partially correct ordering.
To perform the diffusion process as a reordering task the same architecture of the work of Giuliari et al.(11) is used.

Therefore a task-specific backbone is used to perform feature extraction. Then a forward diffusion process is used to shuffle the data and an Attention-based Graph Neural Network (GNN) is used to learn to restore the original positions.

5.3 Methodology

In this section, we summarize the basic steps of our approach.
Each video is divided into clips. Then, a 3D CNN is used to extract features for the clips.
The original position of each clip and the corresponding feature are inserted into a graph structure, where each node represents a single clip and contains the original position of the clip and the corresponding feature. The clips are subsequently shuffled through a diffusion process that gradually introduces noise into the original positions. During the reverse process, an Attention-based Graph Neural Network is trained to estimate the noise to inject. The model then uses the noise to estimate the correct position of each clip.
The main elements of the proposed approach are shown in figure 5.1.
The reordering process is a self-supervised task used to acquire a useful representation

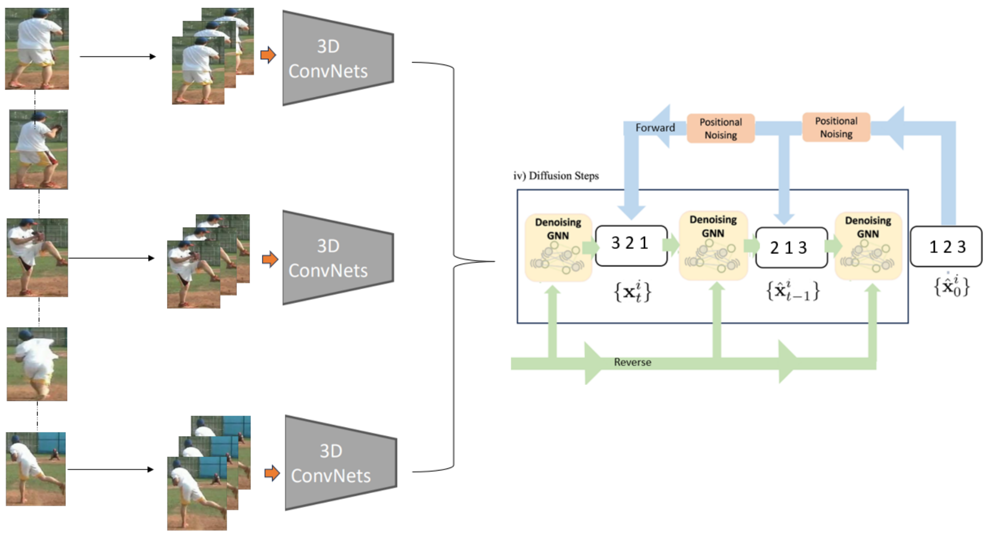


Figura 5.1: Proposed approach. Image adapted from (11) and from (13)
.

of the input videos. Indeed, to correctly reorder the clips of a video, it is necessary to acquire a representation that highlights the differences and similarities of the different phases within a video. The usefulness of the learned representation was therefore tested on several downstream tasks. Specifically, the trained 3D CNNs are fine-tuned on an action classification tasks at video level. The learned 3D CNNs are also used in the same way to perform a phase classification task at frame level.

5.3.1 Input

An interval of m frames is utilized to sample the clips uniformly from the video.
The clips are compelled to be non-overlapping to prevent the entire framework from performing the task by comparing less significant attributes like texture and color. A straightforward comparison of frame pixels can accomplish the task in extreme circumstances where the clips overlap by one frame.
Clips are selected from the beginning of the video. A graph structure, denoted as G, is used as input, whereby each node corresponds to an input point and $\left[\mathbf{x}_t^i; \mathbf{h}^i\right]^\top$ is assigned as node features, where $\mathbf{h}^i$ is derived from the corresponding clip using a pre-trained backbone and $\mathbf{x}_t^i$ corresponds to the original position of the corresponding clip.

5.3.2 Feature extraction

3D CNNs are used to extract features for each clip.
A 3D CNN, or 3D Convolutional Neural Network, is a deep learning model used for processing three-dimensional data.
Unlike traditional 2D CNNs that are commonly used for image analysis, 3D CNNs are designed to work with volumetric data, which has three dimensions, typically representing space and time. They are particularly useful in applications where the data has a temporal and spatial component, such as video analysis.
As shown in figure 2.10, the same 3D CNN is used for all clips in one tuple. Specifically, the C3D network introduced by Tran et al.(40) is employed to perform feature extraction .
Three-dimensional convolutional neural networks (3D CNNs) are particularly suitable to extract feature from clips due to their ability to effectively capture and represent video data's spatial and temporal characteristics.
The C3D network consists of eight stacked convolutional layers, with five pooling layers interspersed between them. All convolution kernels have dimensions of 3×3×3. More details are shown in figure 5.2.
The parameters of stride and padding are modified according to the downstream task tested. In fact, in the case of a video-level action classification, it is not necessary to maintain the same original input size in terms of the number of frames per clip because a single representation from the entire clip is sufficient to perform the task. If, on the other hand, a phase classification is performed, it is necessary to maintain a representation for each frame within the clip. Therefore the stride and padding parameters must be modified accordingly.

5.3.3 Forward and reverse process

Based on the work of Ho et al.(8), the forward process is characterized as a stationary Markov chain that introduces Gaussian noise to the initial location $\mathbf{x}_0^k$ of each input, denoted as $\mathbf{x}^k$, following a Gaussian distribution. During the time interval $t \in [0, T]$, the variance β_t is adjusted based on a linear scheduler. Additionally, the function $q\left(\mathbf{x}_t \mid \mathbf{x}_0\right)$ is defined as:

$$q\left(\mathbf{x}_t^i \mid \mathbf{x}_0^i\right) = \mathcal{N}\left(\mathbf{x}_t^i; \sqrt{\bar{\alpha}_t}\mathbf{x}_0^i, \left(1 - \bar{\alpha}_t\right)\mathbf{I}\right),$$

where $\alpha_t = 1 - \beta_t, \bar{\alpha}_t = \prod_{s=1}^t \alpha_s$.
The noisy location at time t, denoted as $\mathbf{x}_t^k$, can be derived using the above formulation

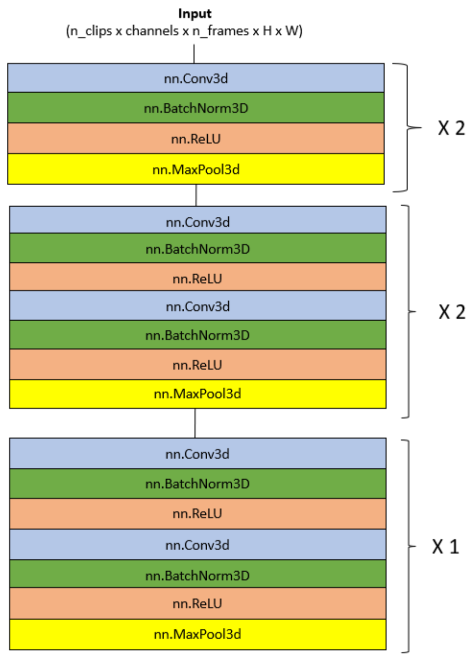


Figura 5.2: C3d network

.

based on the initial position $\mathbf{x}_0^k$.
The reverse process is employed to obtain the accurate position for each data point by utilizing the noisy positions $\mathbf{x}_t^i$ and element features $\mathbf{h}^i$.
The DDIM (41) algorithm is used and the value of $\hat{\mathbf{x}}_{t-1}$ is selected through sampling as:

$$\begin{aligned}\hat{\mathbf{x}}_{t-1} =&\sqrt{\bar{\alpha}_{t-1}}\left(\frac{\mathbf{x}_t-\sqrt{1-\bar{\alpha}_t}\epsilon_\theta\left(\mathbf{x}_t,\mathbf{t},\mathbf{h}\right)}{\bar{\alpha}_t}\right)\\&+\sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\cdot\epsilon_\theta\left(\mathbf{x}_t,\mathbf{t},\mathbf{h}\right)+\sigma_t\epsilon.\end{aligned}$$

Where the function $\epsilon_\theta\left(\mathbf{x}_t,\mathbf{t},\mathbf{h}\right)$ represents the estimated noise that needs to be eliminated from $\mathbf{x}_t$ in order to retrieve the estimated value $\hat{\mathbf{x}}_{t-1}$. The variable t denotes a vector embedding learned for the specific timestep.
In the above formula, the superscript "i" is excluded since the network functions on all elements concurrently in the form of a graph.
DDIM introduces the parameter σ_t to regulate the stochastic sampling process. To ensure determinism in the sampling process, the value of σ_t is set to 0, as there is only one valid arrangement for the ordering task.

5.3.4 Loss function

The estimated noise is used to predict $\mathbf{x}_0$. The Huber loss is used to train the model to predict the correct position of each element.

The Huber loss function is defined as follows:

$$L = \begin{cases} 0.5\left(\hat{\mathbf{x}_0} - \mathbf{x}_0\right)^2/\delta & \text{if } |\hat{\mathbf{x}_0} - \mathbf{x}_0| < \delta \\ |\hat{\mathbf{x}_0} - \mathbf{x}_0\,| - 0.5 * \delta, & \text{otherwise} \end{cases}$$

The Huber loss provides a compromise between the mean squared error (MSE) loss, which is sensitive to outliers, and the mean absolute error (MAE) loss, which is less sensitive to outliers. By adjusting the value of δ, you can control the trade-off between robustness to outliers and the precision of the loss function. By adjusting the value of δ, you can control the trade-off between robustness to outliers and the precision of the loss function. Smaller values of δ make the loss more like MSE, while larger values make it more like MAE.

5.3.5 Network architecture

An Attention-based Graph Neural Network is used to learn the reverse diffusion process and thus estimate the noise to be added in a given time step to reproduce the original position of the clips.
Figure 5.3 shows the network's architecture.
The network comprises a stack of four Graph Transformers layers. Such Graph Transformers use the attention mechanism introduced by Shi et al.(36) on top of a graph structure to control the information gathered from neighboring nodes. Then, a Multilayer Perceptron (MLP) is used to predict the positions of individual nodes from the information extracted from the Attention-based Graph Neural Network.

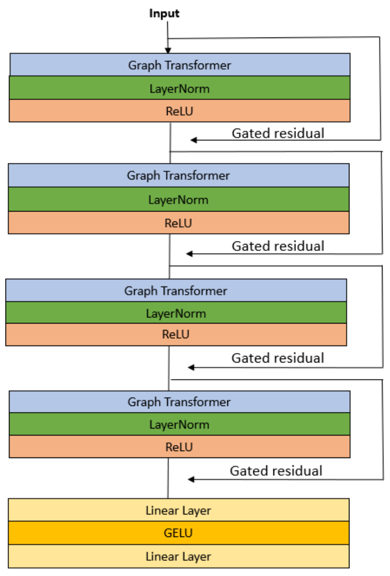


Figura 5.3: Attention-based Graph Neural network's architecture

.

Chapter 6

Experiments

6.1 Overview

This chapter investigates whether the diffusion process is useful in the reordering task. Additionally, it investigates whether the representations learned during the reordering task are meaningful.
The effectiveness of these representations was investigated in various downstream tasks through different experiments. In particular, the usefulness of the learned representation was investigated using transfer learning. Therefore, the weights of the model trained in the reordering task were extracted to initialize a model to be tested in different downstream tasks.
The performance of the pre-trained model was compared with that of the model trained from scratch in an action classification task and a phase classification task.
This chapter describes the implementation details and the experimental setup.

6.2 Reordering Task

6.2.1 Data

Two datasets are used to perform this task: PennAction dataset (14) and UCF101 dataset (15).
PennAction dataset contains 2326 video sequences of 15 different actions. For each video frame, the annotation consists of a 2D keypoint position for thirteen human body joints, their corresponding visibilities, and camera viewpoints.
This dataset was chosen because people are engaged in sporting activity. Therefore, a lot of dynamicity of movement guarantees a significant difference in distinct frames or clips, even if taken at a relatively small interval. This significant difference makes the reordering task easier to perform even with a small interval between clips.

UCF101 is a data set of realistic action videos taken from YouTube with 101 action categories. It contains 13320 videos, with an average clip duration of 1500 seconds.
Five categories can be distinguished from the action categories: 1) Human-Object Interaction, 2) Body- Motion Only, 3) Human-human interaction, 4) Playing Musical Instruments, and 5) Sports.
With 13,320 videos from 101 action categories, UCF101 has large diversity regarding actions and it offers a wide variances in viewpoint, object appearance and pose, camera

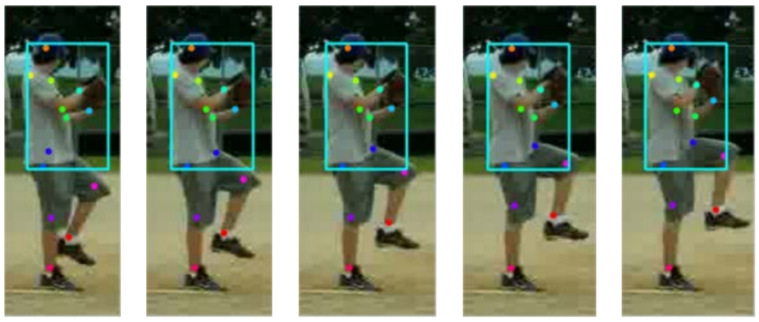


Figura 6.1: An example from PennAction dataset. Image taken from Zhang et al. (14).
.

movements, cluttered background and lighting.

6.2.2 Implementation details

For the PennAction dataset, four frames are sampled for each clip, with a four-frame interval between each clip. The sampling process continues until the video finishes or until less than four frames are left to be sampled. In this way, the number of clips for each video changes according to the length of the original video.
The human bounding box is used to crop each frame of the clips to eliminate background noise and focus on the action performed.
For the UCF101 dataset, the same protocol of Xu et al.(6) is followed. Therefore, three clips are sampled from each video. Sixteen frames are sampled for each clip, with a eight-frame interval between each clip.
Both the backbone and the positional diffusion are trained. All the weights are randomly initialized. All the weights have been updated during the training process using the back-propagation technique.
During training, the AdaFactor optimizer was used with the default values in PyTorch. A batch size equal to 8 was used to train the model. The model was trained for 150 epochs.
The model is validated every ten epochs on the remaining validation split. The validation is not carried out after every epoch but only after certain checkpoints, mainly because the sampling step is quite expensive.
The performance in the reordering task of positional diffusion was compared with that of an attention-guide graph neural network in which no diffusion process is implemented. The model used is the same, but in the second case, the input is not corrupted by adding noise stepwise. In this case, the model does not have to learn the reverse process by predicting the noise to be added at each step but receives as input an array of zeros from which it directly predicts the correct position of each frame. The same method used for the diffusion method is used to train the model.

6.2.3 Evaluation metrics

To evaluate the performance in the reordering task, the following metrics are used:

- *Accuracy*: is the percentage of correctly predicted clip positions. It is calculated as the average of the correct clip position for each tuple of clips.

- *Kendall's Tau*: measures the correlation between the ground-truth orders of clips and the predicted ones. It is a coefficient representing the degree of concordance between two columns of ranked data. Let $(x_1, y_1), \ldots, (x_n, y_n)$ be a set of observations of the joint random variables X and Y, such that all the values of (x_i) and (y_i) are unique. Any pair of observations (x_i, y_i) and (x_j, y_j),where $i < j$, are said to be concordant if the sort order of (x_i, x_j) and (y_i, y_j) agree,i.e. if either both $x_i > x_j$ and $y_i > y_j$ or $x_j > x_i$ and $y_j > y_i$; otherwise they are said to be discordant. The Kendall's Tau coefficient is defined as follows:

$$\tau = 1 - \left(2(\# \text{ Inversions }) \binom{K}{2}^{-1} \right)$$

where K is the number of clips in a tuple and #Inversions is the number of discordant pairs. This metric was introduced to consider the partial ordering of clips. Indeed, partially correct orders are not evaluated using accuracy as a metric. If the clips were sorted oppositely to the original order, or if the order was correct but only shifted one position from the correct positions, the accuracy would still be as low as in the case of a completely wrong sorting.

6.3 Downstream task

The ordering task is used as a pre-task in self-supervised learning.
The usefulness of the representation learned during the ordering task is tested in several downstream tasks. The backbone previously trained in the pre-task is used as a model for each downstream task. Therefore, the weights of the backbone trained in the reordering task were extracted to initialize the model.
A fully connected layer with softmax on top of it was appended as in (6). Only the fully connected layers are randomly initialized; other layers are initialized from the corresponding self-supervised training.
The performance of the pre-trained model was compared with that of the model trained from scratch.

6.3.1 Phase classification

Data

The Penn Action dataset was used to perform this task.
Phase labels are taken from Dwibebi et al.(36). The phase is the time interval between two key events, and every frame in that time has the same phase label.

Implementation details

The reordering task was performed as described in the previous section. Subsequently, the weights of the backbone trained using the reordering task were extracted to initialize the model. The weights of the last fully connected layer are randomly initialized.

The backbone was fine-tuned on a phase classification task for 50 epochs.
All the weights have been updated during the training process using the back-propagation algorithm. During training, the AdaFactor optimizer was used with the default values in PyTorch. A batch size of 8 was employed to train the model.
The model is validated every ten epochs on the remaining validation split. The validation is not carried out after every epoch but only after certain checkpoints, mainly because the sampling step is quite expensive.
The performance of the pre-trained model was then compared with that of the model trained from scratch using the same training procedure. The backbone was initialized with random weights and trained for 50 epochs in a phase classification task.

Evaluation metrics

To evaluate the performance in the phase classification task, the following metric is used:

- *Accuracy*: is the percentage of correctly predicted phase. It is calculated as the average of the correct predicted phase for each frame of tuples of clips.

6.3.2 Action classification task

Data

The UCF101 dataset, as in the work of Xu et al.(6), was used to perform the action classification task.

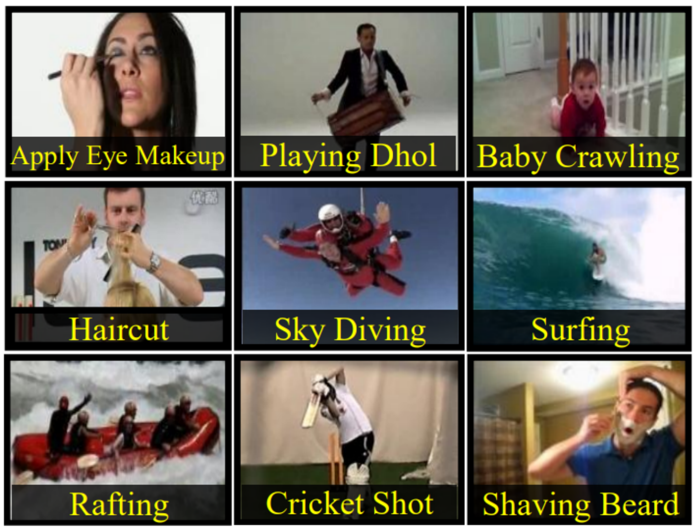


Figura 6.2: Examples from UCF101 dataset. Image taken from Soomro et al. (15).
.

Implementation details

The reordering task was performed as described in the previous section. Then, the weights of the backbone trained in the reordering task were extracted to initialize the model. The last fully connected layer is randomly initialized.
The method from (6) is followed to obtain the action recognition result for a video.Therefore ,ten clips are sampled from each video. The C3D backbone is used to predict the action for each clip, and all predictions are then averaged to obtain the video prediction.
The model is fine-tuned on an action classification task for 150 epochs as in (6). Backpropagation has been used to update all weights during the training phase. The AdaFactor optimizer was utilized during training using the default parameters in Pytorch. Eight batches were used to train the model.
On the remaining validation split, the model is validated every ten epochs. Due to the high cost of the sample process, validation is done only after specific checkpoints rather than after each epoch.
Next, the performance of the pre-trained model was compared with that of the model trained from scratch using the same training procedure.

6.4 Results

6.4.1 Results on the reordering task

Table 6.1 and Table 6.2 show the results of the reordering task. In this task, a video is divided into clips, then the clips are shuffled, and the goal is to predict the correct order of the clips.
As seen from Table 6.1 and Table 6.2, the diffusion process guarantees a performance improvement. However, this improvement is only present in the PennAction dataset. When using the PennAction dataset, the clips to be reordered depend on the length of the video but are, on average, about ten. In contrast, in the UCF101, following the protocol of Xu et al.(6), three clips are reordered at a time. This suggests that the difference depends on the task's difficulty level, and diffusion is useful only in the case of more complex tasks.

Tabella 6.1: Results in the reordering task with and without diffusion process using PennAction dataset

	Accuracy	**Kendall's Tau**
With diffusion	**47.74**	**0.59**
Without diffusion	43.42	0.58

Tabella 6.2: Results in the reordering task with and without diffusion process using UCF101 dataset.

	Accuracy	**Kendall's Tau**
With diffusion	40.54	0.03
Without diffusion	**46.28**	**0.15**

6.4.2 Results on downstream tasks

Results on the phase classification task

Tabella 6.3: Results on the phase classification task.

	Phase acc.	**Ordering acc.**	**Kendall's Tau**
With diffusion	**66.42**	**47.74**	**0.59**
Without diffusion	64.35	43.42	0.58

Tabella 6.4: Results on the phase classification task. Comparison with the model from scratch

	Phase acc.
Best pre-trained model	66.42
From scratch	**67.09**

Table 6.3 shows the results of the phase classification task. In the phase classification task, the weights of the 3D CNN trained in the reordering task were extracted to initialize the model. Then, the network is finetuned on a phase classification task.
As seen from Table 6.3 there seems to be a correlation between the performance obtained in the reordering task and the phase classification task. Introducing the diffusion process improves the reordering process by 3.32 % and the phase classification task by 2.17 %.

In this preliminary study, as seen from Table 6.4, training a model from scratch results in better performance. This suggests that the representations learned during the reordering task are not useful for performing well in a phase classification task.

Results on the action classification task

Tabella 6.5: Results on the action classification task

	Action acc.	**Ordering acc.**	**Kendall's Tau**
With diffusion	14.94	40.54	0.03
Without diffusion	**17.71**	**46.28**	**0.15**

Tabella 6.6: Results on the action classification task. Comparison with the model from scratch

	Action acc.
Best pre-trained model	17.71
From scratch	**21.83**

Table 6.5 shows the results of the action classification task. In the action classification task, the weights of the 3D CNN trained in the reordering task were extracted to initialize the model. Then, the network is finetuned on an action classification task.

Table 6.5 shows a possible correlation between the pre-ordering task and the action classification task. The attention-guided graph neural network without diffusion process is the model that performs best in both the action classification task, with an accuracy of 46.28 %, and in the action classification task, with an accuracy of 17.71 %.

However, in this preliminary study, the information learned during the reordering does not seem useful for the downstream task. The model trained from scratch performs better than the model using the weights learned during the reordering task. In particular, the model trained from scratch achieves an accuracy of 21.83 % in the action classification task, while the transform achieves an accuracy of 17.71 % and the diffusion model an accuracy of 14.94 %.

Chapter 7

Conclusions

In this work, a method to perform a self-supervised task was used. Self-supervised tasks could overcome traditional supervised learning methods limitations, as described in Chapter 2.
As a self-supervised task, a reordering task is used. Thanks to this task, it was possible to exploit the correlations between temporally close frames to learn a meaningful representation of the input videos.
To perform the reordering task, each video was divided into clips. Clips are then shuffled, and a model is used to reconstruct the correct order of the clips.
A clip reordering task was used rather than a frame reordering task because reordering individual frames could be an ambiguous task as, in some cases, it is difficult to identify the correct order due to the potential viability of different action directions. Furthermore, a clip-level task allows learning a representation at the frame and video levels, providing more exhaustive information for possible downstream tasks.
The reordering task was performed as a regression task. Using the reordering task as a regression rather than a classification task reduces the complexity of the task as the number of items to be reordered increases. In addition, this also allowed the use of more detailed metrics to evaluate the reordering task that also took partially correct ordering into account.
A diffusion model was used to perform the reordering task, as previous work (11) has shown that diffusion models can be useful in reordering tasks in a 2D space. A diffusion model was, therefore, used in a reordering task where the only dimension present was the temporal one.
The experiments carried out in this work investigate the usefulness of the diffusion process in the reordering task and the meaningfulness of the learned representation. In particular, two downstream tasks were used: an action classification task and a phase classification task.
The results show that the introduction of the diffusion process appears to be functional for the reordering task only when the task is complex. Experiments highlight that diffusion is useful only when a variable number of clips greater or equal to ten must be reordered. In contrast, the introduction of the diffusion model does not lead to an improvement in performance when there are only three elements to be reordered. It would be interesting to investigate this further by analyzing the model's performance with or without diffusion, as the number of elements to be reordered varies.
Furthermore, these preliminary studies show that the reordering task doesn't help to learn useful representations for the downstream tasks investigated. It would be

interesting to investigate other downstream tasks in future work. Another element to be investigated in future work is how the proportion of labeled data may change the performance, as the self-supervised task is likely to be useful especially when labeled data are insufficient to train a network from scratch properly. Another thing that would be interesting would be to vary the reordering task in part by using individual frames rather than clips to have a direct frame-to-frame comparison, which, when using downstream tasks at the frame level, could allow for improved performance. Finally, future work could investigate whether features extracted at later stages in the reordering process could prove useful in the downstream tasks under investigation.

Bibliography

[1] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III 14*, pp. 649–666, Springer, 2016.

[2] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *Proceedings of the IEEE international conference on computer vision*, pp. 1422–1430, 2015.

[3] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," *arXiv preprint arXiv:1803.07728*, 2018.

[4] T. H. Trinh, M.-T. Luong, and Q. V. Le, "Selfie: Self-supervised pretraining for image embedding," *arXiv preprint arXiv:1906.02940*, 2019.

[5] H.-Y. Lee, J.-B. Huang, M. Singh, and M.-H. Yang, "Unsupervised representation learning by sorting sequences," in *Proceedings of the IEEE international conference on computer vision*, pp. 667–676, 2017.

[6] D. Xu, J. Xiao, Z. Zhao, J. Shao, D. Xie, and Y. Zhuang, "Self-supervised spatiotemporal learning via video clip order prediction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10334–10343, 2019.

[7] D. Dwibedi, Y. Aytar, J. Tompson, P. Sermanet, and A. Zisserman, "Temporal cycle-consistency learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1801–1810, 2019.

[8] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[9] A. Lugmayr, M. Danelljan, A. Romero, F. Yu, R. Timofte, and L. Van Gool, "Repaint: Inpainting using denoising diffusion probabilistic models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11461–11471, 2022.

[10] D. Baranchuk, I. Rubachev, A. Voynov, V. Khrulkov, and A. Babenko, "Label-efficient semantic segmentation with diffusion models," *arXiv preprint arXiv:2112.03126*, 2021.

[11] F. Giuliari, G. Scarpellini, S. James, Y. Wang, and A. Del Bue, "Positional diffusion: Ordering unordered sets with diffusion probabilistic models," *arXiv preprint arXiv:2303.11120*, 2023.

[12] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, *et al.*, “Graph attention networks,” *stat*, vol. 1050, no. 20, pp. 10–48550, 2017.

[13] W. Liu, B. Tekin, H. Coskun, V. Vineet, P. Fua, and M. Pollefeys, “Learning to align sequential actions in the wild,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2181–2191, 2022.

[14] W. Zhang, M. Zhu, and K. G. Derpanis, “From actemes to action: A strongly-supervised representation for detailed action understanding,” in *Proceedings of the IEEE international conference on computer vision*, pp. 2248–2255, 2013.

[15] K. Soomro, A. R. Zamir, and M. Shah, “Ucf101: A dataset of 101 human actions classes from videos in the wild,” *arXiv preprint arXiv:1212.0402*, 2012.

[16] Y. M. Asano, C. Rupprecht, and A. Vedaldi, “Self-labelling via simultaneous clustering and representation learning,” *arXiv preprint arXiv:1911.05371*, 2019.

[17] D. H. Ballard, “Modular learning in neural networks,” in *Proceedings of the sixth National Conference on artificial intelligence-volume 1*, pp. 279–284, 1987.

[18] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” *Advances in neural information processing systems*, vol. 27, 2014.

[19] M. Noroozi and P. Favaro, “Unsupervised learning of visual representations by solving jigsaw puzzles,” in *European conference on computer vision*, pp. 69–84, Springer, 2016.

[20] G. Camporese, E. Izzo, and L. Ballan, “Where are my neighbors? exploiting patches relations in self-supervised vision transformer,” *arXiv preprint arXiv:2206.00481*, 2022.

[21] M. Noroozi, H. Pirsiavash, and P. Favaro, “Representation learning by learning to count,” in *Proceedings of the IEEE international conference on computer vision*, pp. 5898–5906, 2017.

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.

[23] I. Misra, C. L. Zitnick, and M. Hebert, “Shuffle and learn: unsupervised learning using temporal order verification,” in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pp. 527–544, Springer, 2016.

[24] P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, S. Levine, and G. Brain, “Time-contrastive networks: Self-supervised learning from video,” in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 1134–1141, IEEE, 2018.

[25] S. Haresh, S. Kumar, H. Coskun, S. N. Syed, A. Konin, Z. Zia, and Q.-H. Tran, “Learning by aligning videos in time,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5548–5558, 2021.

[26] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series," in *Proceedings of the 3rd international conference on knowledge discovery and data mining*, pp. 359–370, 1994.

[27] R. W. Conners and C. A. Harlow, "A theoretical comparison of texture algorithms," *IEEE transactions on pattern analysis and machine intelligence*, no. 3, pp. 204–222, 1980.

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

[29] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *International conference on machine learning*, pp. 2256–2265, PMLR, 2015.

[30] W. Feller, "Fluctuation theory of recurrent events," *Transactions of the American Mathematical Society*, vol. 67, no. 1, pp. 98–119, 1949.

[31] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," *Advances in neural information processing systems*, vol. 34, pp. 8780–8794, 2021.

[32] C. Saharia, J. Ho, W. Chan, T. Salimans, D. J. Fleet, and M. Norouzi, "Image super-resolution via iterative refinement," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 4, pp. 4713–4726, 2022.

[33] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever, and M. Chen, "Glide: Towards photorealistic image generation and editing with text-guided diffusion models," *arXiv preprint arXiv:2112.10741*, 2021.

[34] S. Mukhopadhyay, M. Gwilliam, V. Agarwal, N. Padmanabhan, A. Swaminathan, S. Hegde, T. Zhou, and A. Shrivastava, "Diffusion models beat gans on image classification," *arXiv preprint arXiv:2307.08702*, 2023.

[35] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

[36] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun, "Masked label prediction: Unified message passing model for semi-supervised classification," *arXiv preprint arXiv:2009.03509*, 2020.

[37] S. Haykin, *Neural networks: a comprehensive foundation.* Prentice Hall PTR, 1998.

[38] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[40] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 4489–4497, 2015.

[41] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," *arXiv preprint arXiv:2010.02502*, 2020.

[42] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.