**ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA**

**DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING**

ARTIFICIAL INTELLIGENCE

MASTER THESIS
in
Computer Vision

ENHANCING A TEXT-SHAPE COHERENCE METRIC BY LEVERAGING CONTRASTIVE LOSSES AND A LARGE-SCALE DATASET

CANDIDATE
Davide Mercanti

SUPERVISOR
Prof. Samuele Salti

CO-SUPERVISOR
Andrea Amaduzzi, PhD.

Academic Year 2022 — 2023
Session III

CONTENTS

INTRODUCTION

Recently, many conditional generation methods for 3D object have been developed. However, the development of such methods is faster than the development of metrics able to evaluate results quality, especially in terms of text-shape coherence.

Most of the widespread metrics are only able to evaluate generative quality and not text-shape coherence, or rely upon 2D renderings of the generated shapes, which makes them computationally expensive and sensitive to rendering parameters.

In this work a coherence metric named CrossCoherence (CC), recently proposed by [Amaduzzi et al., 2023], which is trained on chairs and tables only, will be analysed and extended to deal with (almost) any kind of shape. This metric works directly with point clouds to get rid of the dependence from rendering processes and leverages cross-attention to target the coherence between text and shape.

Our analysis will show that (a) the application of contrastive training losses can enhance CC results and that (b) increasing data volume and diversity significantly enhance generalization capabilities of the model and quality of the results – as it is often the case with deep architectures – and enables CC to be used as a general coherence metric.

The **code** used for the experiments in this work (part of which is experimental and not intended to be used for production) is available at:

`https://github.com/nonci/cross_coherence_GPT2S`

`https://github.com/nonci/cross_coherence_objaverse`

CHAPTER 1 TEXT-TO-SHAPE GENERATION

1.1 Introduction

Two main paradigms to generate 3D shapes from natural text have proven to be successful.

A first one, whose underlying ideas are the same used for text-conditioned image generation, directly relies on paired text-shape datasets. For the training of such models to be effective, texts has to be as descriptive and specific as possible, hence the scarcity of such datasets.

A second approach leverages existing models trained to align text to images. These systems are usually based on optimizing NeRF (NEural Radiance Field, [Gao et al., 2023; Mildenhall et al., 2020]) or other continuous functions and have the major drawbacks of extreme slowness and lack of interpretability, since the shape is encoded in the network weights, rather than in a mesh or in a point cloud.

In this chapter I am going to analyse text-to-shape (T2S) systems with the double purpose of getting to know what later will be to evaluate and of taking inspiration to build the evaluating architecture. Before diving into a description of the two approaches, I will give an account of the kinds of data we need (or we have) and, by the end of chapter, the datasets that are currently available for T2S generation with such data.

1.2 Input data

3D shapes can be represented in different ways, the most common being the following.

Volumetric data: voxels and octrees

These methods represents shapes by partitioning the space with a 3D-grid. The information associated to every grid cell is called **voxel** and it usually encodes occupancy or colour of the cell. The position described by the voxel is not encoded within the voxel itself, but can be derived from the voxel's position in the data structure, given that each cell of the grid has to be represented in its position.

Voxelization, ie. the representation or conversion of a shape by using voxels, can be seen as the straightforward extension of the common image representation that adopts *pixels*, thus enabling simple extensions of 2D computer vision techniques to 3D scenarios.

Important limitations of this format are:

- The fact that both occupied and non-occupied cells must be represented leads to high inefficiency, especially when resolution increases (that is, the size of each cell decreases).
- Voxelization does not preserve intrinsic properties of the shape[1], nor the smoothness of the surface.

A more efficient representation employs **octrees** [Meagher, 1980; Tatarchenko et al., 2017], that are 8-ary trees obtained by partitioning the space occupied by the shape into a grid of variable-size cubic cells. The procedure starts considering as a cell the whole volume (or each cell of a coarse voxelization), then the cell is recursively divided into eight *octants* (sub-cells) if the value of the function we want to encode, like occupancy or colour, varies inside the cell. Otherwise, the cell will be a terminal node in the tree.

In this way, it is possible to drastically reduce the memory occupation of a voxelized shape, that is still in the order of the object surface, without further loss. This representation, however, still suffers from the second problem of the two we have previously listed.

Implicit representations from 2D images

3D data can be represented from a series of 2D images rendered from different point of views. This procedure allows leveraging the existing feature extraction methods based on images, while reducing problems related to noise, occlusion and non-lambertianess of the surface.

[1]Coordinates are usually opposed to intrinsic properties, ie. the ones that are independent from the adopted coordinate system. Descriptors derived from the coordinates, or from point-of-view-dependent data such as renderings, are referred to as *extrinsic descriptors*, otherwise they are called *intrinsic descriptors*.

Specifically, the aim is to learn a function, usually encoded in a NeRF model, able to represent the whole shape and capable of good generalization to other 3D shapes. This can be obtained by jointly optimizing the functions modelling each rendering.
Than, the shape is represented by the learned function itself (i.e., it's encoded in the neural network weights).

Even if this method is sensitive to the "number of renderings" hyperparameter, it remains one of the most effective.

Meshes

Another possible representation for a 3D shapes is a mesh or polyhedron, namely a collection of vertices (point coordinates), edges (couples of vertices) and faces (closed set of edges). Often, attributes such as textures are associated to faces and edges. Commonly, faces are triangles or quadrilaterals.

For example, a single face would be encoded as:

```
V = [[0.00495, 0.27978, -0.00617],
     [4.1411e-05, 0.27987, -0.00617],
     [4.0850e-05, 0.27935, -0.00648]]
E = [[0, 1], [1, 2], [2, 0]]
F = [[0, 1, 2]]
```

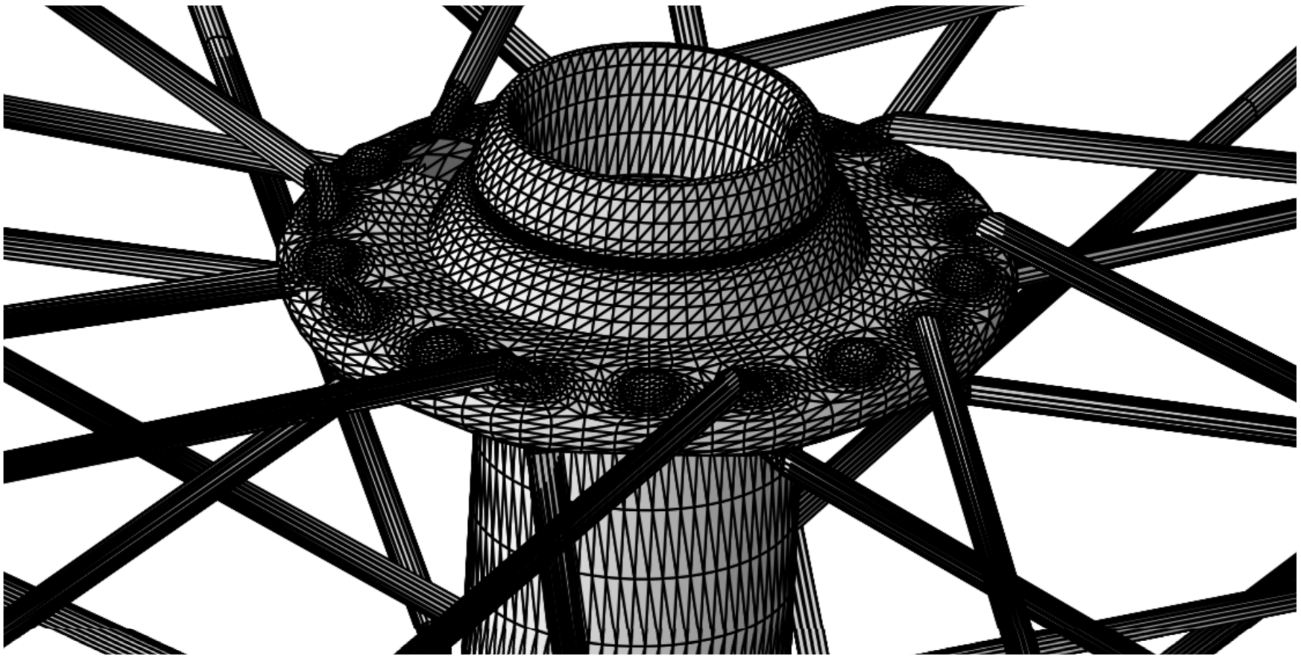


Figure 1.1 – *Rendering of a triangular mesh of a bike hub, displaying contours of each face. Small inconsistencies at the spoke holes are present in the original data.*

This format, despite being widespread for rendering, gives rise to non-trivial problems, for example when dealing with deformations of the space. Therefore, more complex representations for meshes have been proposed (e.g. [Lipman et al., 2005]).

Point clouds

This representation approximates a shape by using an unordered set of points, each one characterized by its coordinates and, optionally, by a set of features, such as point colour.

In the following experiments we use extensively a representation in which each point is encoded by 3 Euclidean coordinates and 3 subsequent RGB colour coordinates. A single-point shape would be represented as tensor

```
[[ 0.1795, -0.0494, -0.4921,  0.0000,  0.4627,  0.7020]]
```

where each colour component is normalized into [0,1].

Due to its lack of structure (as opposed to meshes, there is no information about connectivity among points), this representation may induce ambiguity about surfaces, especially with a low number of points (see fig. 1.2, for example).

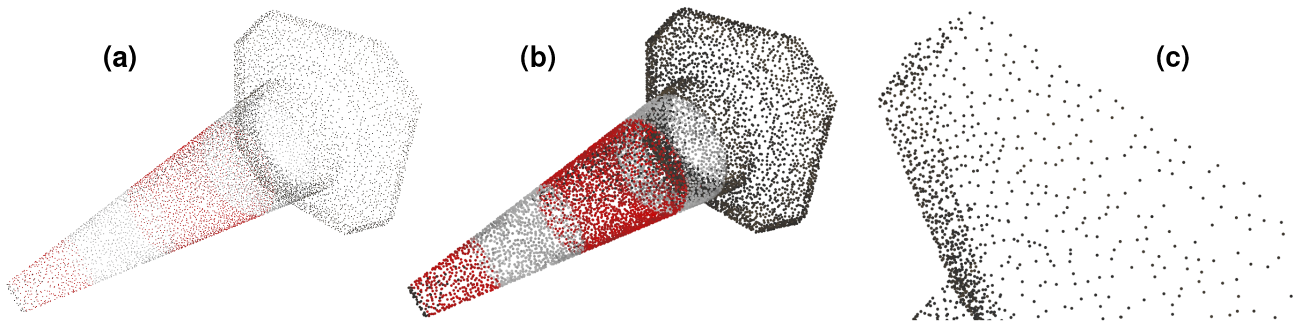


Figure 1.2 – *Rendering of a point cloud of a cone. (a) and (b) represent the whole shape, with different sizes of the circular marker used to display each point; (c) shows a part of the cloud. Note that inferring the cone surfaces is not trivial in the last case.*

1.3 Architectures

As anticipated in § 1.1, we can discriminate between two different approaches to conditioned cloud generation: one directly based on textual supervision of 3D shapes and another one that leverages text-to-image (T2I) models, or embeddings learned trough such models.

In this section we are going to be briefly describe the former category[2], more aligned to our purposes, focusing on the aspects relevant for the subsequent parts.

1.3.1 Based on Supervised Learning

Text2Shape

A seminal work that introduced the “supervised approach” for text-to-shape generation is Text2Shape ([Chen et al., 2018]). The authors rely on a subset of *ShapeNet*, augmented with text descriptions they collected (see § 1.4.1), other than a procedurally-generated dataset of 3D simple shapes, to learn joint cross-modal[3] embeddings.

To learn semantically meaningful embeddings (that are tested for both retrieval and generative tasks), the authors jointly train in a end-to-end fashion a text and a shape encoder based on convolutions and FC layers (plus a Gated Recurrent Unit for the text encoder and a pooling layer for the shape encoder).

The loss used in this work uses both the concepts of *learning by association* and *metric learning*, combining them in a multi-objective function. At first, given a matrix of shape embeddings S and one of text embeddings T, the matrix M is computed, defined by:

$M_{ij} = T_i \cdot S_j$

Then, M is row-wise softmax-normalized:

$P_{ij}^{TS} = e^{M_{ij}} / \sum_k e^{M_{ik}}$

to obtain at each row i the probability of text description i to be associated to each shape. Similarly, a matrix P^{ST} is computed starting from M^T, expressing the probability of each shape i to be associated to each text description.

Then, the following so-called *round-trip probability* matrices are computed:

$P_{ij}^{TST} = (P^{TS} P^{ST})_{ij}$

in which for each text i the probability distribution in P_{ij}^{TST} is imposed to be “uniform over the descriptions j which are similar to description i” by the means of a cross-entropy loss L_R^{TST} between the distribution P^{TST} and the target distribution (uniform across similar shapes).

[2]Although some of the architectures that will be presented still leverage renderings or CLIP embedding, too.

[3]*Cross-modality* refers to relations that link text and shape (the two *modalities*).

This constraint allows the model to learn cross-modal many-to-many associations and it is summed to a simpler component (the use of which has found to be beneficial) to get a cross-modal association loss L^{TST} (refer to the article for details, especially figure 3b).

Similarly, a loss L^{STS} is enforced.

The final loss described in the paper combines these losses with two other components, aimed at imposing metric constraints, each one with a different modality as anchor. Particularly, a loss L_{ML}^{TT} uses text embedding as an anchor and works within-modality, while another loss L_{ML}^{TS} uses shape embedding as an anchor and works across modalities (see fig. 3c in [Chen et al., 2018]).

$L_{total} = L^{TST} + L^{STS} + \gamma(L_{ML}^{TT} + L_{ML}^{TS})$

The encoders trained with this compound loss have been effectively used by the authors for retrieval by nearest-neighbourhood in the embedding space and for shape generation, in association with a GAN.

Other Works

Since T2S, some other works have followed the supervised approach, among which:

- **ShapeCrafter** [Fu et al., 2023], based on the idea of progressively refining a shape by adding more precise information, as humans tend to do when describing 3D shapes. Text descriptions are taken from Text2Shape++, a dataset made of hierarchical descriptions built for the purpose. The architecture is trained to align text features generated by a fine-tuned BERT encoder and shape features generated by a VAE from the shape. The architecture is recurrent and has to merge, at each step, the latent features coming from the previous step (that can be decoded as a chape) to the ones derived from the additional text information.

- **Shap·E** [Jun and Nichol, 2023], that uses two separately-trained parts: an encoder trained to produce the parameters of an implicit representation of known 3D shapes, in this case the weights of a MLP[4], and a diffusion model, conditioned on images or text descriptions, trained on the latent representation produced by the encoder. The encoder inputs are a 3D point cloud, CLIP embeddings and renderings and leverages cross-attention prior to a Transformer architecture to learn the relations in input data.

- **Towards Implicit Text-Guided 3D Shape Generation** [Liu et al., 2022], in which the authors

[4]This MLP works “as both a NeRF and a signed texture field (STF)”.

build a network made of (a) a shape encoder, in the form of a CNN trained as a part of an auto-encoder; (b) a text encoder based on BERT to obtain text embeddings; (c) a Word-Level Spatial Transformer (WLST), that is a module leveraging cross-attention to correlate between embedded text tokens and shape local features and (d) a spatial-aware decoder that is initialized with the weights of the decoder originally used in the shape AE and that has to reconstruct the input shape given the global features produced by *(a)* and *(b)* and the local features produced by the WLST. This architecture is trained with a compound loss, a part of which is called *cyclic consistency loss*. This consistency loss enforces proximity between the global shape features and the ones obtained by re-encoding the shape obtained from *(d)*.

1.4 Datasets

1.4.1 Text2Shape

ShapeNet dataset has been introduced in [Chang et al., 2015] as a collection of 3D CAD models with hierarchical semantic labelling according to WordNet [CITE] taxonomy. **Text2Shape (T2S)** was introduced in [Chen et al., 2018], and it is based on two ShapeNet categories, *chairs* (8,447 instances) and *tables* (6,591 instances), augmented with 75,344 natural language descriptions (thus, configuring a one-to-many association) collected via crowdsourcing.

[Amaduzzi et al., 2023] report high variability in quality and descriptiveness of such captions, that include errors and irrelevant details. Particularly, the lack of details makes this dataset unsuitable for generative tasks.

Another variant of this dataset, **Text2Shape++** (presented in [Fu et al., 2023]), contains incremental descriptions built upon Text2Shape ones, thus suffering from the same weaknesses.

1.4.2 GPT2Shape and HST

Due to the limited quality of T2S, [Amaduzzi et al., 2023] introduced a new automatically-generated dataset. The authors note that "enough details about a shape are usually present in Text2Shape if information from multiple prompts are merged". So, they propose to generate more informative descriptions by prompting a LLM (OpenAI GPT-3 model *davinci*, in this case) with the original descriptions plus a request to rephrase, result of an empirical *prompt engineering* process:

"Describe geometry, shape, color and other key features in less than 40 words, in 5 different ways"

Then, LLM output can be used to form a new dataset, called **GPT2Shape** (GPT2S). The authors decided to keep the same amount of shapes (15,032) and text descriptions (75,358) of T2S and proved the increased informativeness of GPT2S by a user study in which participants were provided with a text description (from T2S or GPT2S) and asked to select one between two shapes, the one that more closely paired with it, or a "cannot decide" option. The two shapes, the *ground truth* and a *distractor*, were selected to be similar by computing their distance in PointNet++ auto-encoder embedding space. Some examples are reported in fig. 1.3.

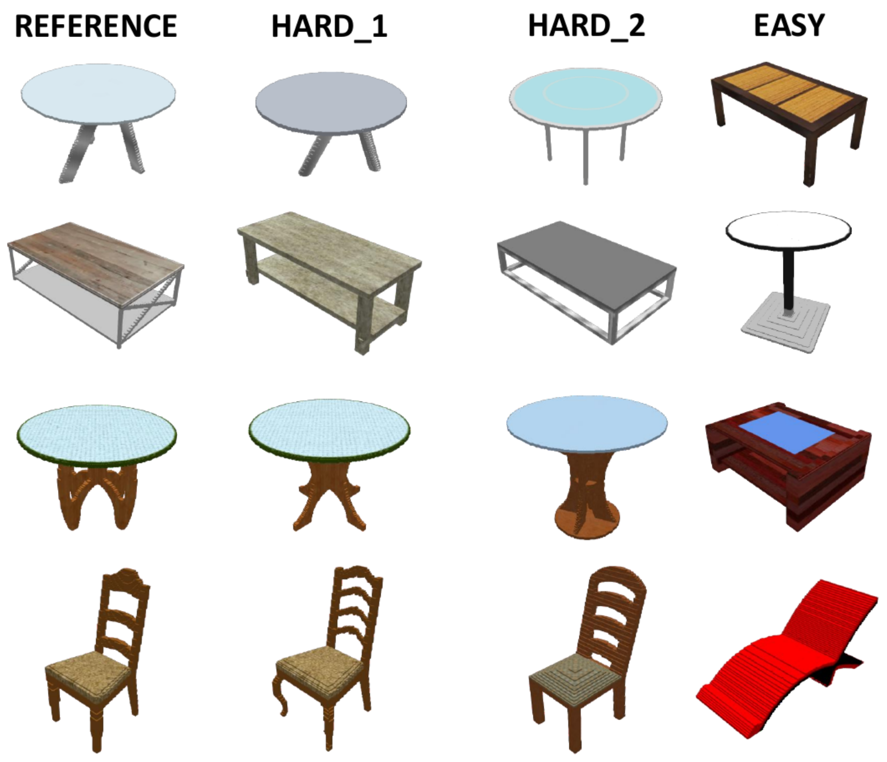


Figure 1.3 – *Shapes from T2S, taken from [Amaduzzi et al., 2023]. "Hard_1" and "Hard_2" refer to the two closest shapes in the embedding space of PointNet++; "Easy" refers to a randomly sampled shape from the easy distractors.*

Results of the same user study were used to assemble a high-quality Human-validated Shape-Text (**HST**) dataset by selecting only the shape-text pairs for which there was full consensus among participants about the right shape-text association and no "cannot decide" answers. Selected pairs were 2,153 and are saved along with the distractor.

1.4.3 Objaverse and Cap3D

A far more comprehensive and realistic dataset of 3D models is Objaverse [Deitke et al., 2022]. **Objaverse 1.0**, used for this study, contains around 800K 3D shapes of (at least) 1.1K *semantic object categories*.

Objaverse has been released with some metadata. For example, metadata associated with shape 8476c4170df24cf5bbe6967222d1a42d are:

```
{ ’uri’: ’https://api.sketchfab.com/v3/models/8476c4170df24cf5bbe6967222d1a42d’,
  ’uid’: ’8476c4170df24cf5bbe6967222d1a42d’,
  ’name’: ’Iain_Dawson_Kew_Road_Formby’,
  ’staffpickedAt’: None,
  ’viewCount’: 4,
  ’likeCount’: 0,
  ’animationCount’: 0,
  ’viewerUrl’: ’https://sketchfab.com/3d-models/8476c4170df24cf5bbe6967222d1a42d’,
  ’embedUrl’: ’https://sketchfab.com/models/8476c4170df24cf5bbe6967222d1a42d/embed’,
  ’commentCount’: 0,
  ’isDownloadable’: True,
  ’publishedAt’: ’2021-03-18T09:36:25.430631’,
  ’tags’: [
    { ’name’: ’stair’,
      ’slug’: ’stair’,
      ’uri’: ’https://api.sketchfab.com/v3/tags/stair’},
    { ’name’: ’staircase’,
      ’slug’: ’staircase’,
      ’uri’: ’https://api.sketchfab.com/v3/tags/staircase’},
    { ’name’: ’staircon’,
      ’slug’: ’staircon’,
      ’uri’: ’https://api.sketchfab.com/v3/tags/staircon’}
  ],
  ’categories’: [],
  ’thumbnails’: { ... },
  ’user’: { ... },
  ’description’: ’http://staircon.com/ <br>Export by <b>Alan Grice Staircase Co. Ltd
   </b> (lic 6391)’,
  ’faceCount’: 14608,
  ’createdAt’: ’2021-03-18T09:31:52.190927’,
  ’vertexCount’: 7309,
  ’isAgeRestricted’: False,
  ’archives’: { ... },
  ’license’: ’by’
}
```

These metadata, however, greatly varies in accuracy and descriptiveness. Tags and description are empty for many of the objects, thereby limiting their use for our purposes.[5]

Recently, however, [Luo et al., 2023] proposed **Cap3D**, an automatic pipeline to caption 3D objects, that they applied to Objaverse 1.0 with performances surpassing that of human annotators.

Their method applies four subsequent phases:

1. Each 3D object is rendered from 8 point of views using Blender;
2. From every image, 5 captions are extracted using BLIP (a method leveraging a LLM to bridge a frozen image encoder and a frozen language model, introduced in [Li et al., 2023]).
3. Cosine similarity on CLIP embeddings (see § 2.2) is used to estimate the coherence between each render and the relative caption. Only the pair with highest similarity is selected[6].
4. Finally, GPT-4 is used to aggregate the 8 captions resulting from previous steps in a single coherent description. GPT-4 is shown to be able to filter out unlikely details while effectively summarizing relevant pieces of information.

Since some technical and ethical filtering is applied to Objaverse, the final dataset is made of 661K pairs. In this work, a subset of this dataset released as `Cap3D_automated_Objaverse_highquality.csv` in [Tiange, 2023] has been used. This file contains 549,922 pairs, obtained by further filtering the dataset.

[5]A subset of 47K shapes uniquely assigned to one of 1156 categories, named Objaverse-LVIS, has been built by the authors. More details can be found in the original paper.

[6]The authors emphasize that using CLIP to evaluate what is produced by BLIP-2 makes sense because the two are somehow complementary, being trained on different data and on a different architecture.

CHAPTER

2 EXISTING METRICS FOR TEXT-SHAPE COHERENCE

2.1 Based on Distances from Ground Truth

Many of the studies that recently explored text-to-shape generation, such as [Chen et al., 2018; Liu et al., 2022; Sanghi et al., 2022], measure text-shape coherence by computing distance measures between the shape generate by the model and the ground-truth shape. This metrics, however, appear inadequate to measure both the visual quality of the generated shape, which does not necessarily correlate with its proximity to the GT shape, and its coherence with the input prompt, for which this correlation is even lower.

2.2 Based on Renderings Only (CLIP)

To overcome the limitations of the methods presented above, it is possible to leverage — again — the already existent models effectively trained to align images with text.

The most interesting metrics of this kind are based on CLIP (Contrastive Language-Image Pre-training), introduced in [Radford et al., 2021], that implements the idea of *contrastive learning* to learn *joint embeddings*, i.e. vectors in a latent space that semantically represent both a shape and a text (they should capture *the concept* shared by the two different representations). Each of the two modalities is encoded with a different encoder, but the two architectures are jointly trained to reach the very same latent space.

More precisely, a Transformer is used as a text encoder and a Vision Transformer (ViT, or alternatively a ResNet network) as an image encoder. Then, the activations of the highest layer of the Transformer corresponding to the *End Of Sequence* token are layer-normalized and linearly projected in the common (multi-modal) embedding space. Similarly, the image features computed by ViT are linearly projected into the same embedding space. Training aims at the alignment of such projections by the means of a contrastive loss.

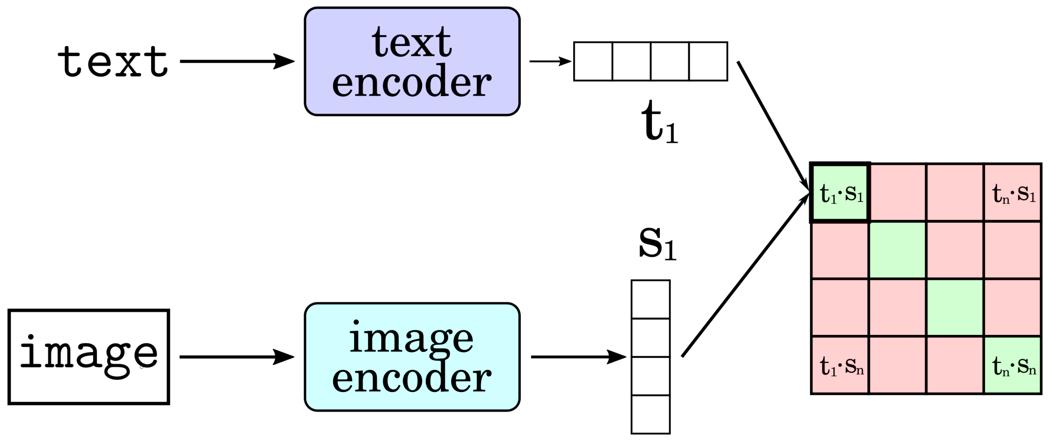


Figure 2.1 – *CLIP training phase. Layer normalization and the linear projection layer between each encoder and the matrix are not represented.*

To align these projections, CLIP loss builds a matrix containing each possible dot product (distance) between text and shape embeddings and tries to align this matrix to the identity (see fig. 2.1) by using dimension-wise cross-entropy losses. A pseudo-implementation of the loss is given in § 3.1.1 and in the original paper.

2.2.1 CLIP Similarity

CLIP Similarity (CLIP-S) has been introduced in [Fu et al., 2023] to evaluate text-to-shape generation. It leverages CLIP embeddings by the following procedure:

1. Twenty views per shape are rendered;
2. A pre-trained CLIP model is used to extract image features from each of the view and text features from the shape caption;
3. The maximum cosine similarity calculated between text features and each image features is normalized in a pre-defined range[1] and taken as a value of CLIP Similarity.

[1]Two pre-determined shape-text pairs are encoded, one highly coherent and the other poorly coherent, and the value of their embeddings defines the range.

This metric, despite being effective in evaluating local and global coherence between the two modalities, is sensitive to rendering parameters. Furthermore, its dependence from rendering means that it is hard, if not impossible, to compare 3D models encoded with different representations (whose renderings will not appear homogeneous; see also [Amaduzzi et al., 2023, § 2]).

2.2.2 CLIP R-Precision

CLIP R-Precision is another metric based on CLIP joint embeddings that has been introduced by [Zhu et al., 2023].

Specifically, it corresponds to the accuracy obtained when using CLIP-S to retrieve the correct caption among a set of distractors (152, in the experiments), given a rendering of the generated shape.

Still, this metric is based on rendering, thus is not able to overcome CLIP-S limitations.

2.3 CrossCoherence

CrossCoherence (CC), introduced by [Amaduzzi et al., 2023], is a metric based on cross-attention. In order to decouple metric and rendering, thereby circumventing the aforementioned sensitivity to rendering parameters, CC exploits cross-attention to directly learn to quantify the correlation between the shape embedding and the embedding of a corresponding textual description.

2.3.1 Overall Architecture

CC takes as inputs a (coloured) point cloud and a text and outputs a value representing the adherence between the two.

The overall architecture is represented in fig. 2.2.

At first, a shape encoder $\mathcal{E}_T$ converts the point cloud into semantically meaningful local embeddings and a text encoder $\mathcal{E}_S$ does the same for text. Particularly, the PC is fed to $\mathcal{E}_S$ after proper normalization to get the shape features $\mathscr{X}_S$, while the input description is tokenized and then fed to the Transformer-like encoder $\mathcal{E}_T$ to get (possibly per-token) text features $\mathscr{X}_T$.

Then, layer normalization is applied to $\mathscr{X}_S$ and $\mathscr{X}_T$ (not represented) and the normalized activations are fed to the cross-attention. It worth noting that the use of layer normalization allows for a fair

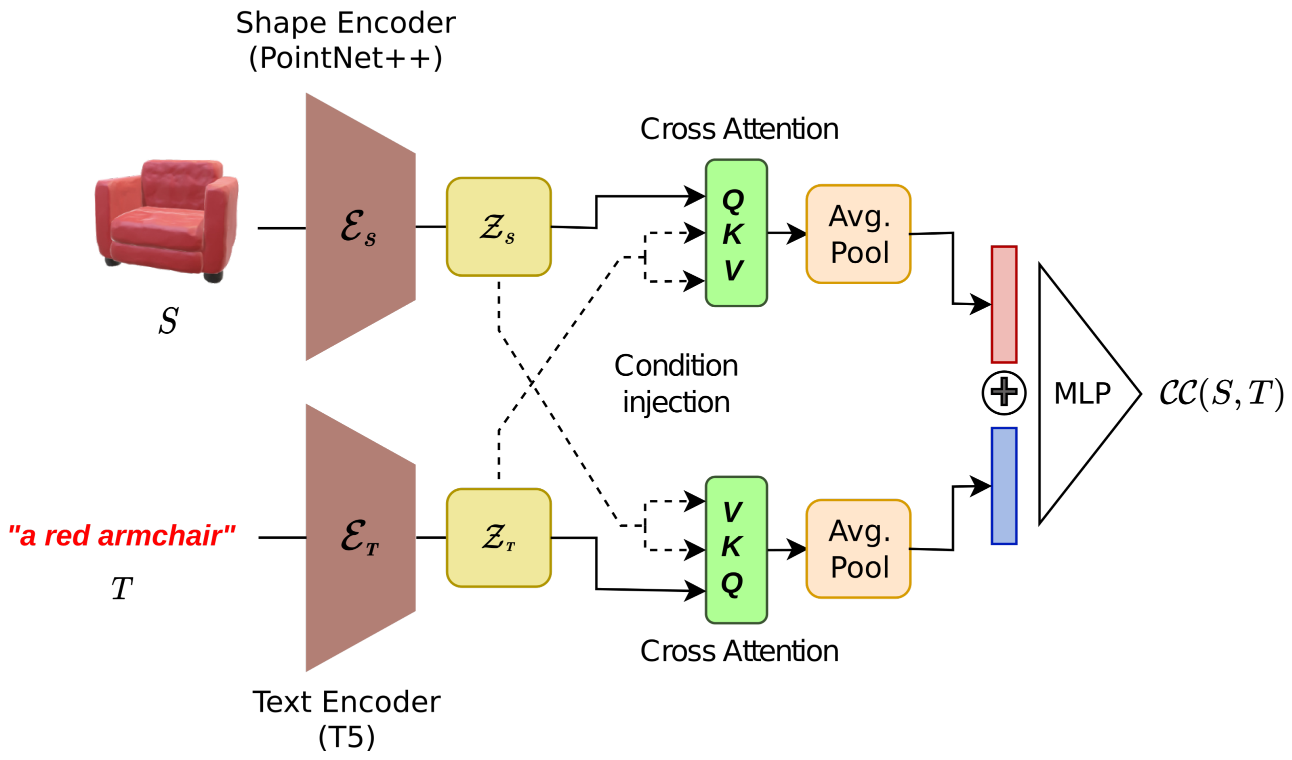


Figure 2.2 – *CrossCoherence architecture. [Adapted from the original paper]*

comparison among different batch sizes in the experiments that will be described.

After the cross-attention layer, that gives to CC the ability to reason about the correlation between shape and text details, the embeddings are average-pooled and fed to a MLP to get a single value representing the correlation score of the text-shape couple.

Dimensionalities of both data and activations are shown in fig. 2.3. Clouds are initially made of 2048 $<x,y,z,R,G,B>$ points, from which 128 features are extracted. Text is tokenized (with a maximum of 77 tokens) and encoded as the activations of the last hidden layer of a Transformer architecture. Later, two symmetric cross-attention layers bring the dimensions of both feature sets to 512 and a pooling operation is performed across the features dimension, to obtain two 1-D vectors. These vectors are finally transformed into a single value by a MLP.

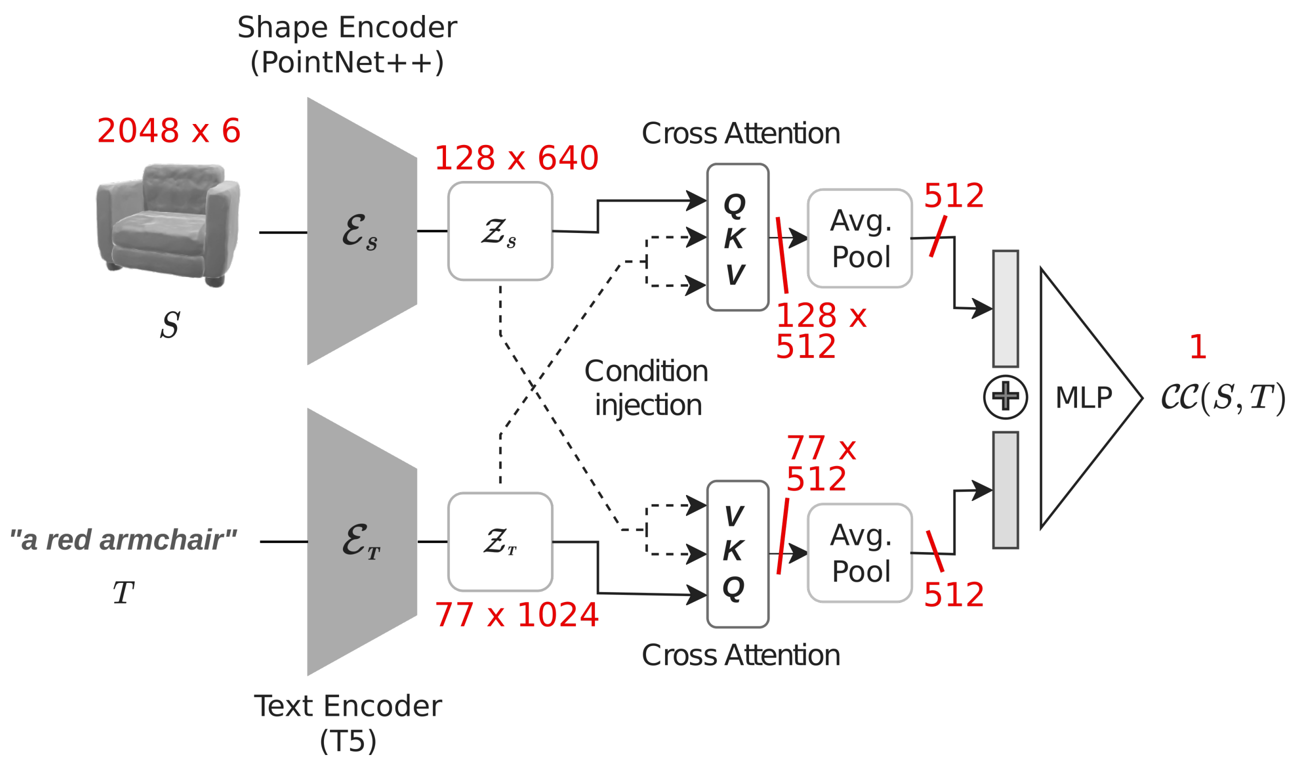


Figure 2.3 – *Data/activations dimensionality at each point in CC architecture (non-batched case).*

2.3.2 Encoders

In the original paper, PointNet++ [CITE] is used as a PC encoder, while T5 [CITE] is used as a text encoder.

PointNet++ is a network based on PointNet [Qi et al., 2017], that aims at finding representative features in geometric point sets, ie. points in Euclidean spaces, whose density is assumed variable across space.

With respect to PointNet, feature extraction is made hierarchical. In particular, each layer of such hierarchy is called *set abstraction layer* and is implemented as a sequence of:

- **Sampling layer**, that uses *farthest point sampling* (FPS) to select a subset of point that will be considered centroids of the local regions. Note that this way of selecting points is data-dependent, differently from a convolutional operator, for example, that is distribution-agnostic.
- **Grouping layer**, that uses either *ball query* or kNN search to output k neighbouring points for each centroid. Note that k is variable when using ball search (since point density varies), while is fixed in the kNN case. As PointNet++ authors emphasize, this makes the former more suitable

for applications that require the local region to be generalizable across space, such as the ones that need local pattern recognition (as a coherence metric does).

- **PointNet layer**, that takes as input the (variable-sized) regions aggregated by the previous layer, translates the coordinates of each point into the local frame (relative to the centroid) and abstracts each region by the centroid itself and a local feature encoding the neighbourhood.
  While in PointNet this abstraction is performed by mini-PointNet, described in [Qi et al., 2017, supplementary, sec. C] (it is essentially a FC network, with a first part shared among points, a max. pooling and another FC part), in PointNet++ the neighbourhood is considered at different scales, so that "each abstraction level extracts multiple scales of local patterns and combine them intelligently [ie., with different weights] according to local point densities".

2.3.3 Cross Attention

It is important to note that CC architecture uses a bilateral cross-attention, i.e. one in which each attention module takes the *query* from the embeddings of a modality and *keys/values* from the embeddings of the other modality, symmetrically.

This, combined with the locality of the embeddings, allows the model to learn which part of the shape is associated to which word and vice-versa, at the same time. These pieces of information are later synthesised by the MLP.

2.4 T^{3}Bench

T^{3}Bench [He et al., 2023] is a recently-introduced metric to evaluate text-to-shape generation. The authors propose to separately evaluate:

- **Generation quality**, by means of renderings and *regional convolutions*;
- **Shape-text alignment**, by means of multi-view captioning to generate a textual prompt that is later evaluate by a LLM in terms of similarity with respect to the ground-truth prompt.

This approach, despite its effectiveness, still relies of rendering techniques. Sensitivity to rendering parameters is somehow mitigated by the high number of different views employed, but still this method suffers from the other criticalities we have mentioned.

CHAPTER

3

EXPERIMENTS

3.1 Enhancing CC using contrastive training

3.1.1 Using CLIP loss

While in the original paper "at training time, the scores are softmax-normalized and a standard cross-entropy loss is used to guide the network to produce higher scores for the correct pair", here we decided to explore the use of contrastive training by using CLIP (Contrastive Language-Image Pre-training), a method already introduced in § 2.2.

Differently from CLIP, the main objective was not to jointly train the encoders, but rather to jointly train the attention mechanisms, other than the final MLP that regresses the coherence score, while the encoders were kept frozen. For this reason, the matrix has to be build after the final MLP, as shown in fig. 3.1.

To apply CLIP in this fashion, some practical problems arise:

1. The use of N point cloud embeddings and N text embeddings increases quadratically the complexity with respect to the train originally proposed for CC, thereby limiting N.

2. T2S includes more than one description for many of the shapes, in the form of multiple shape-text rows with the same *shape id*. While this is irrelevant when training with pairs (since one of the rows is selected), when dealing with N shapes and N texts (that makes N^2 CC values) it is advisable that shapes do not repeat and no more than one text is correctly associated with each shape; otherwise the matrix should be interpreted differently and the right/wrong ratio would become unpredictable.

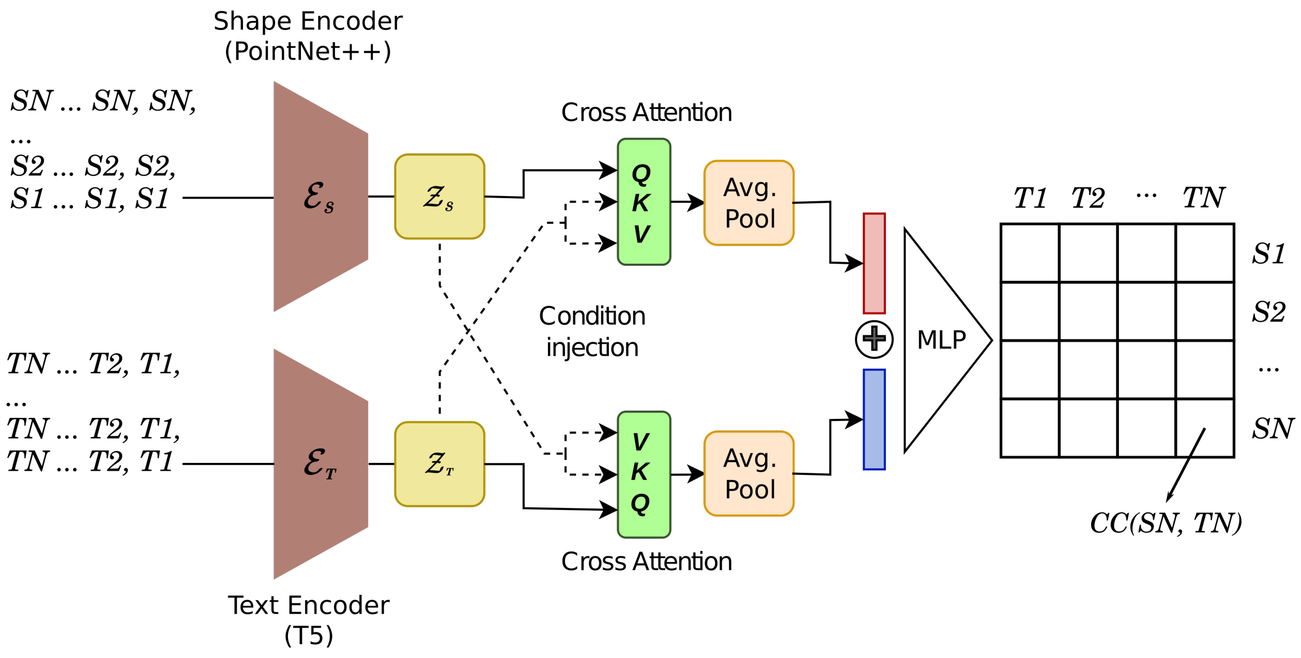


Figure 3.1 – *Application of contrastive loss to CC. Note that input and matrix permutation is not displayed here, but it is necessary to avoid the net learning the ground truth diagonal matrix.*

3. The fact that attention already works with a batch dimension means that all the text-shape couples in the matrix has to be fed to the model as a single batch, rather than in a NxN fashion. This entails the repetition of each shape and of each text N times, to get N^2 couples, before the model is called.

   From now on, we will refer to N as the number of shapes/texts we use to form a batch and to an *actual batch size* $B = N^2$ as the actual number of text-shape couples we use each batch.

4. Either point clouds or texts has to be permuted, to avoid the network trivially learning the ground-truth matrix, and the resulting matrix has to be interpreted according to the permutation.

   This has to be considered along with the previous point.

While the first problem comes with the method itself and can only be mitigated, the other ones need to be solved, as it will be described in the next paragraphs.

Batch formation

As anticipated, we decided to have unique shapes inside each batch. Pre-compute batches with unique shapes, despite its efficiency, would potentially impede the regularization effect induced by non-deterministically sampling the whole dataset at each epoch, as we verified empirically.

A better solution is to create a map from each shape to a list of the corresponding texts found in the dataset:

```
{
  '1bcd9c3fe6c9087e593ebeeedbff73b': [
    'A white chair with metal legs and a square backrest.',
    'A grey chair with a tall backrest and metal legs.',
    ...
  ],
  '5a52b62f564eb7e117b431cae0dd70ed': [
    'Sleek white chair with metal arms and legs.',
    ...
  ]
}
```

Each list of texts should be shuffled each time this data structure is re-created (usually once per epoch).

Then, calling this dictionary S, to get the couples to form a batch the following procedure can be applied:

```
while |S| >= batch_size
  sampled_ids ← randomly sample batch_size shape ids from S
  selected_texts ← []
  for each id in sampled_ids
    selected_text.prepend(S[id].pop())
    if |S[id]| = 0
      S.remove(id)
  yield sampled_ids, selected_texts
```

Note that this is suboptimal, since due to the variable size of the lists associated to shapes, some descriptions may remain unused. Nonetheless, this is not likely to hinder training with relatively small batch sizes, thanks to the random choice of both the shapes that will form the batch and the description of each shape.

Specifically, the maximum length of the lists is 16 and this value rapidly decreases (the mode is 5), so that, for example, the number of discarded text descriptions each epoch is around 0.625% in the worst case, with batch size 32. Problems could still arise with high batch sizes.

The actual implementation of what described so far is available at [CITE], with the difference that, instead of just texts, the list associated to each *shape id* contains texts and cloud embeddings too.

Once the ground truth couples are available, the next step is the proper repetition of the clouds to produce the N^2 couples that will represent the actual batch. This is obtained by repeating in two different ways shape and text embeddings:

```
T1, T2, ..., TN  -->  T1, T2, ..., TN, T1, T2, ..., TN, T1, T2, ..., TN
S1, S2, ..., SN  -->  S1, S1, ..., S1, S2, S2, ..., S2, SN, SN, ..., SN
```

Batch permutation

The final step is to randomly permute the input elements, so that the network does not learn to produce the diagonal ground-truth matrix. This step is only required during training.

If the permutation happens after embeddings repetition, it has to respect batch boundaries and to be the same inside each batch. Since the final matrix is interpreted following raster order, permuting inside groups of *batch size* elements corresponds to a permutation of the columns (text embeddings).

It worth noting that, since the output is interpreted as a matrix, a permutation of the columns will change the index of the right text-shape association (the *label*) both on the columns and on the rows; for example, when applying the following permutation to columns[1]

$$\begin{matrix} 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{matrix} \xrightarrow{\sigma_{col}=3021} \begin{matrix} -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 \\ -1 & -1 & 1 & -1 \\ 1 & -1 & -1 & -1 \end{matrix}$$

the labels of each column correspond to the permutation itself (3021), while row labels can be easily found by considering the indices of the original labels (0123) in σ_{col}: $\sigma_{row} = 1320$, or equivalently to find that permutation σ_{row} s.t. $\sigma_{row}(0123) = \sigma_{col}$.

Training and loss details

After some hyperparameters tuning, a effective training has been performed using Adam optimizer, with a initial lr of $5 * 10^{-4}$, kept constant for a few epochs[2] and later decayed exponentially with $\gamma = 0.95$ (see fig. 3.2). This set of trainings will be referred to as *train 1*.

[1]The actual values produced by CC are different from ± 1, but this will serve us to illustrate Sigmoid loss, too.
[2]The actual number of epochs before decay has been set to 5 for all the experiments, although the best number in terms of learning speed should vary depending on the problem.

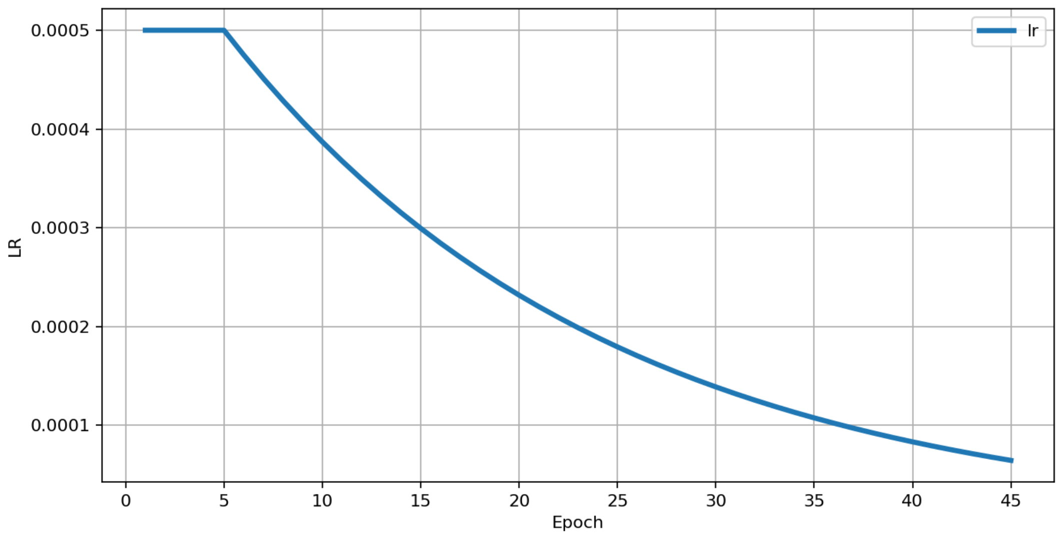


Figure 3.2 – *Learning rate decay policy.*

Following [Radford et al., 2021], CLIP loss has been implemented as the mean of cross entropy on each axis:

```
loss_txt = cross_entropy_loss(logits, col_labels, axis=0)
loss_shp = cross_entropy_loss(logits, row_labels, axis=1)
loss = (loss_shp + loss_txt)/2
```

where `col_labels` and `row_labels` correspond to σ_{col} and σ_{row}.

Training has been performed using a train-validation split to T2S (around 90%-10%).

3.1.2 CLIP loss results

To monitor training dynamics, at each epoch the following metrics inspired by the bi-linearity of CLIP loss have been computed:

- **shape N-accuracy** ($Nacc_S$, when using NxN matrices), the fraction of shapes that are associated to the correct text.
- **text N-accuracy** ($Nacc_T$, when using NxN matrices), the fraction of texts that are associated to the correct shape.
- **(full) N-accuracy** ($Nacc$, when using NxN matrices), the fraction of correspondent row-column pairs for which the model correctly associates shape and text. Ignoring permutation, this corre-

sponds to:

```
labels ← [0..N]
count_nonzero((max_index(logits,dim=0)==labels) ∧  (max_index(logits,dim=1)==labels))
```

It worth noting that this is more restrictive than what the former two accuracies require and it is related to the model capability of learning the fact that the matrix represents N 1 to N associations.

It is clear that each of these metrics strongly depends from N, that determines the difficulty of the problem. For this reason, that would otherwise prevent a fair comparison, N has been fixed to 2 in validation. As an additional (validation-only) metric, the same protocol used to compute accuracy in [Amaduzzi et al., 2023] on HST was used, on the same dataset. This metric will be referred to as **HST accuracy**.

HST accuracy protocol works as follows. The model is fed with triplets, a text and two shapes, and has to determine which of the shapes the text refers to. The accuracy is the fraction of correctly classified triplets. Since this is independent from N, by using HST accuracy it is possible to compare numerically both the enhancements with respect of the original CC and the relative performances of the models tested. Nonetheless, it is important to observe that HST does not exhibit the exact same data distribution of T2S, used for training and most of the validation, since only a part of its shapes are taken from T2S.

Another important factor to be considered is the relative difference that exists among the objects that are included in the matrix: the larger this difference, the easier the problem. This factor will be critical when using shapes from categories other than chairs and tables. For the sake of this paragraphs, it is enough to observe that having both chairs and tables makes the problem easier, while restricting ourselves to one of the two categories makes the problem more subtle and leads sooner to over-fitting.

Dependence from N

Due to limited GPU capacity, experiments were limited to N=32; particularly the following batch sizes have been explored for CLIP-CC with *train 1* protocol:

- N=2 (45 epochs);
- N=8 (45 epochs);
- N=16 (45 epochs);

- N=32 (16 epochs only).

Looking at validation 2-accuracy for the full training, a trend can be recognized (fig. 3.3). It is clear that the size of the matrix used for training influences both convergence speed and accuracy.

In terms of generalization capabilities, the huge difference is between 2 by 2 case and the bigger sizes. Interestingly, very good results can be already obtained with N=8 (64 combinations are simultaneously optimized).

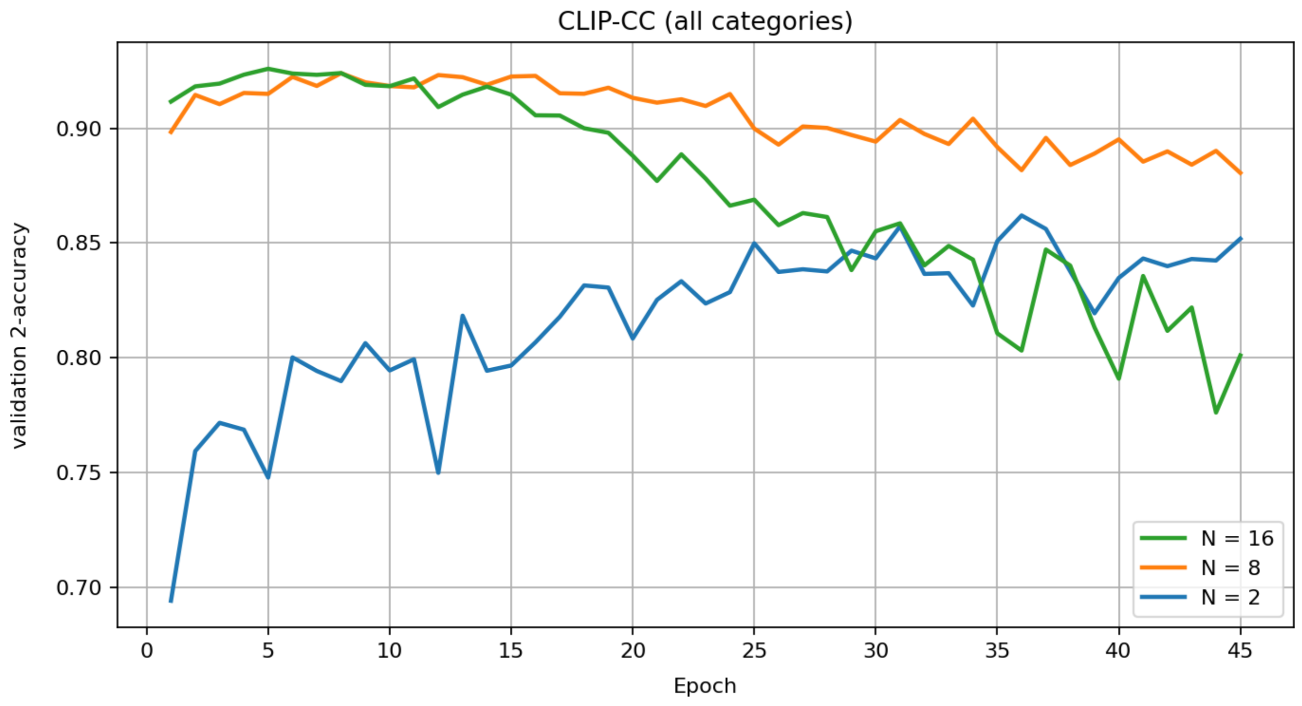


Figure 3.3 – *Validation 2-accuracy of CLIP-CC for different batch sizes.*

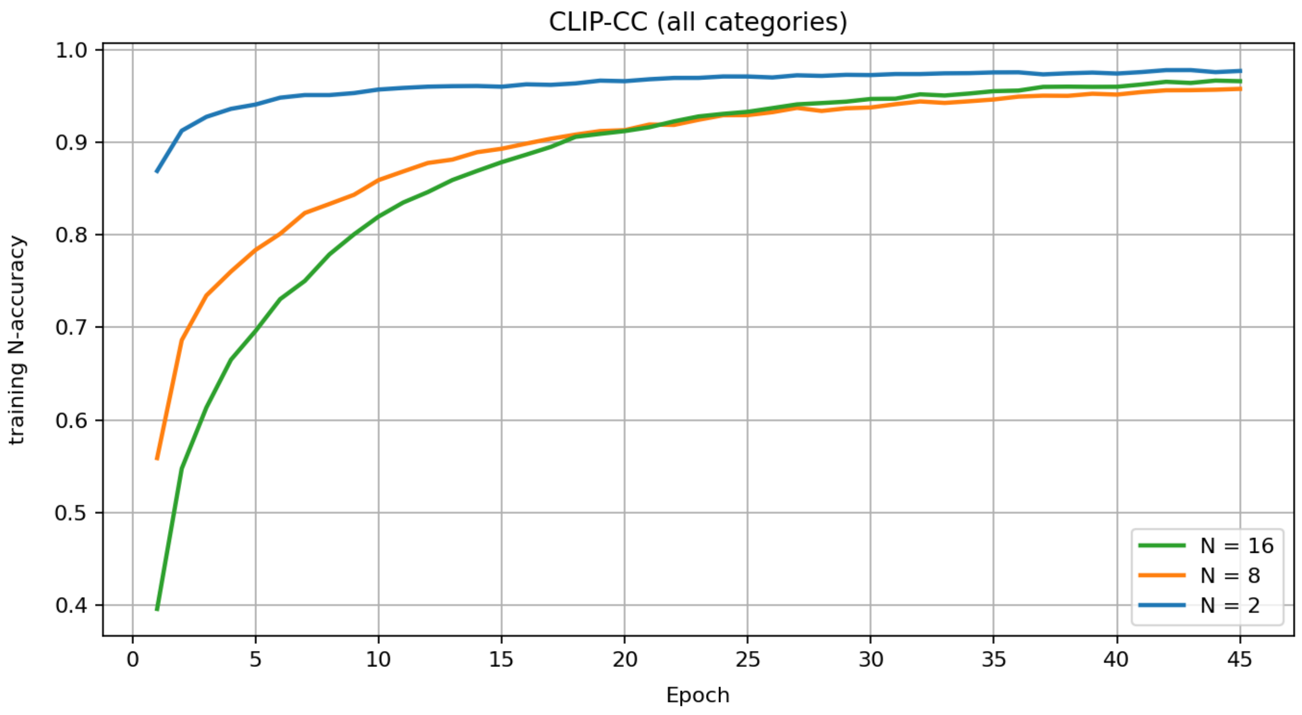


Figure 3.4 – *Training N-accuracy of CLIP-CC for different batch sizes.*

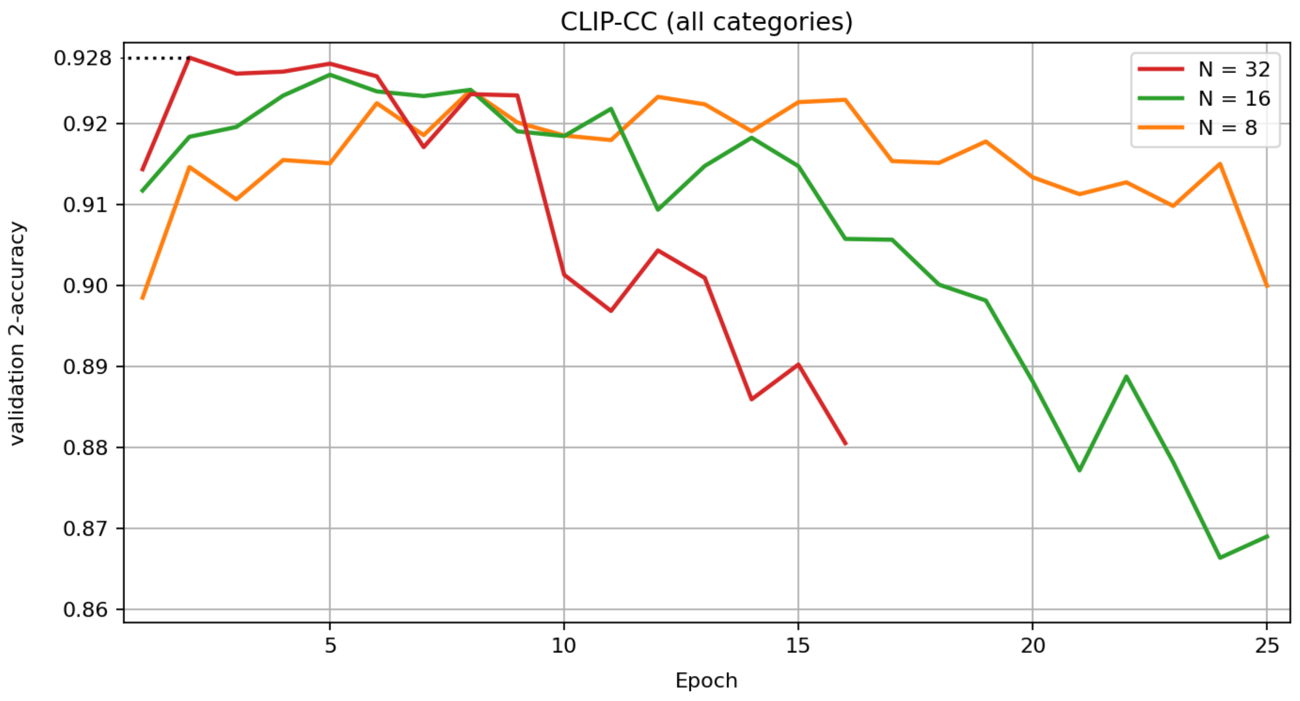


Figure 3.5 – *Validation 2-accuracy of CLIP-CC for different batch sizes (zoomed), including N=32.*

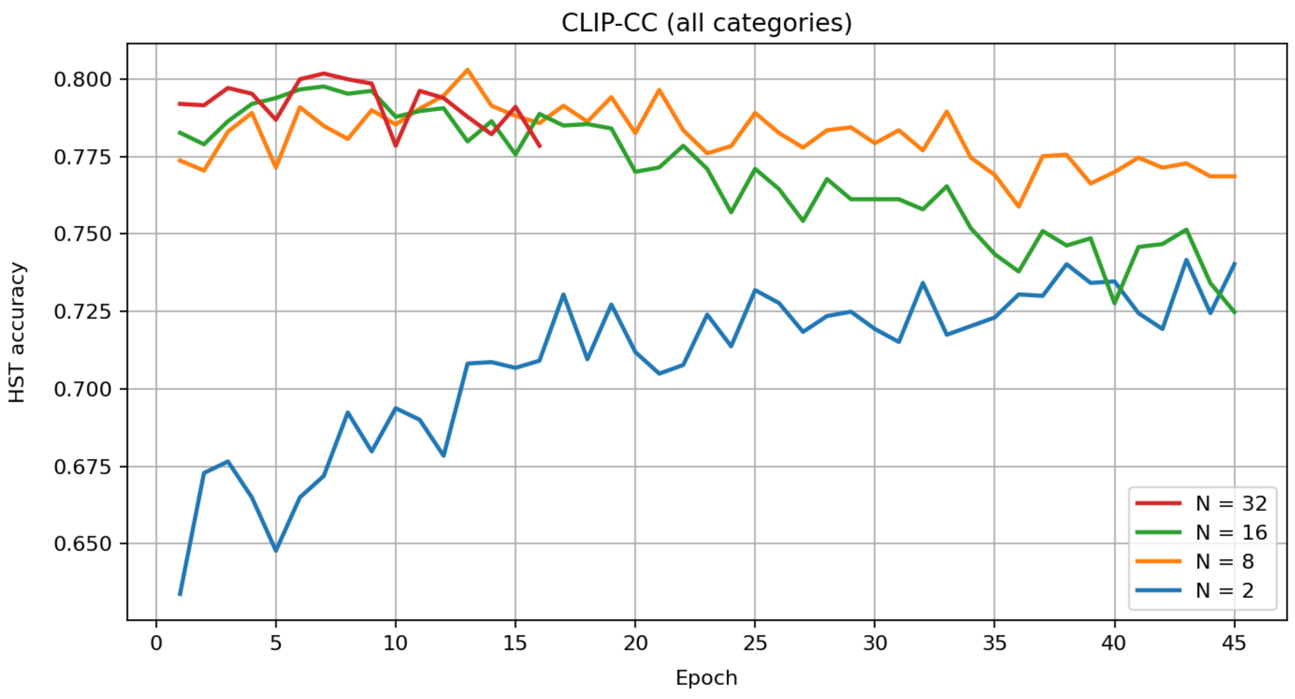


Figure 3.6 – *HST accuracy of CLIP-CC for different batch sizes.*

Another interesting aspect emerging when comparing validation accuracy with training accuracy (fig. 3.4) is that increasing N dramatically increases overfitting: while training accuracy remains high as expected[3], validation accuracy drops very fast.

The training performed with N=32 confirms these considerations (see fig. 3.5), showing the higher validation 2-accuracy (by a small margin) and immediate over-fitting.

It is interesting to see the performances of these models on HST by the means of HST accuracy (fig. 3.6). Apparently, a modest over-fitting on the original distribution does not hurt, or even boosts, generalization capability over this new distribution.

Considerations about contrastive training

By looking at both text and shape accuracies (fig. 3.7), the trend appears always very similar, for both training and validation data. Interestingly, however, shape accuracies are slightly higher, with the relative difference decreasing when the batch size increases.

In light of the findings, we could conjecture that:

[3]By looking at training N-accuracy, note that high accuracies can be obtained, in the end, despite the problem being much harder, confirming the efficacy of training.

- The fact that shape accuracy is higher
  - could reflect the inferior descriptiveness of the text with respect to shapes;
  - may suggest that the problem of associating a shape to the right text is easier than the opposite problem or
  - could simply reflect the fact that the current architecture uses different kinds and sizes of embedding for texts and shapes, thereby introducing an initial asymmetry (prior to bilateral attention).
- The reducing difference may be due to contrastive learning itself, specifically from the different number of examples of the other modality the network optimizes for at each backpropagation step for each text (or shape); optimizing jointly for the two objectives and with a sufficient intra-modality variability the inferior performances coming from texts may reduce.

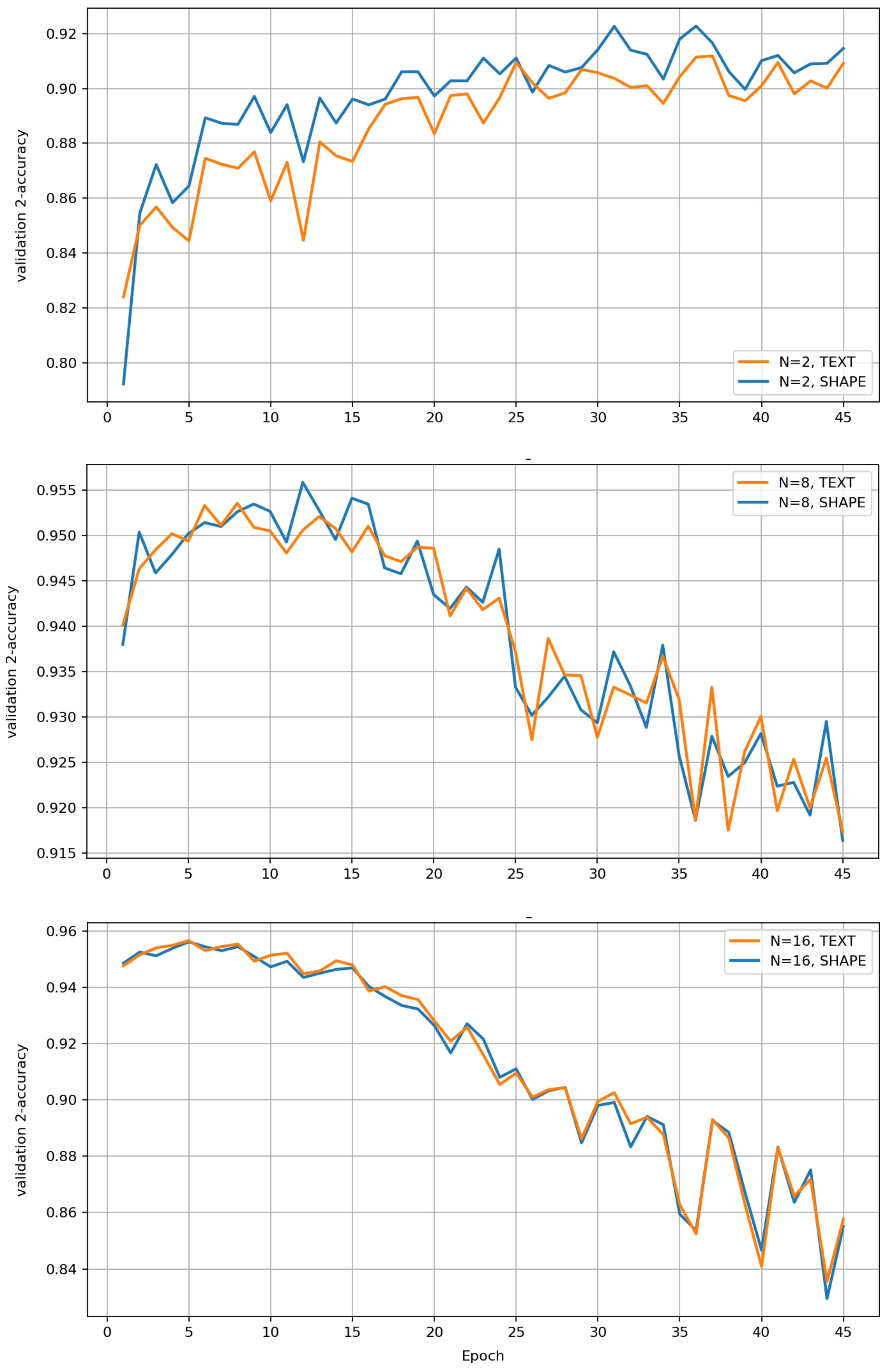


Figure 3.7 – *Validation shape and text 2-accuracies during training with different batch sizes.*

3.1.3 Using Sigmoid Loss

Another interesting possibility to apply contrastive learning in the CC scenario is to use Sigmoid Loss, as proposed for image-text learning by [Zhai et al., 2023].

This loss function still has to be applied to the usual N^2 matrix, but it allows to disentangle the loss value from the batch size.

Indicating with **x** our matrix and with **y** the smoothed one-hot encoding of the corresponding ground-truth labels, the cross-entropy would be computed as

$$L_{row}(\mathbf{x},\mathbf{y}) = -\frac{1}{N}\sum_{n=0}^{N-1}\sum_{c=0}^{N-1} \mathbf{y}_{nc}\, log\left(\frac{e^{\mathbf{x}_{nc}}}{\sum_{i=0}^{N-1} e^{\mathbf{x}_{ni}}}\right)$$

(Similarly for L_{col})

$$L_{CE}(\mathbf{x},\mathbf{y}) = \frac{1}{2}(L_{row} + L_{col})$$

Instead, Sigmoid Loss computes, on the whole matrix

$$L_{\sigma}(\mathbf{x},\mathbf{y}) = -\frac{1}{N}\sum_{n=0}^{N-1}\sum_{c=0}^{N-1} log\left(\frac{1}{1+e^{\mathbf{y}_{nc}(-t\,\mathbf{x}_{nc}+b)}}\right)$$

where **y** is no more a one-hot encoding, but contains a 1 for the right pairings and -1 otherwise (as depicted in the matrix on page 26), and t and b are learnable parameters.

Similarly to what is suggested in the original paper, this has been implemented in torch starting from:

```
# b and t are optimizable parameters
logits = logits * exp(t) + b

# matrix of -1, with diagonal 1s:
labels = 2 * eye(N) - ones(N)

# ... if training, permute labels  ...

loss = -sum(log_sigmoid(labels * logits)) / N
```

Note that, while L_{CE} formulation does include a normalization sum inside logarithm, L_{σ} treats each logit of the matrix independently, so that the problem is reduced to binary classification of each possible pair, gaining efficiency.

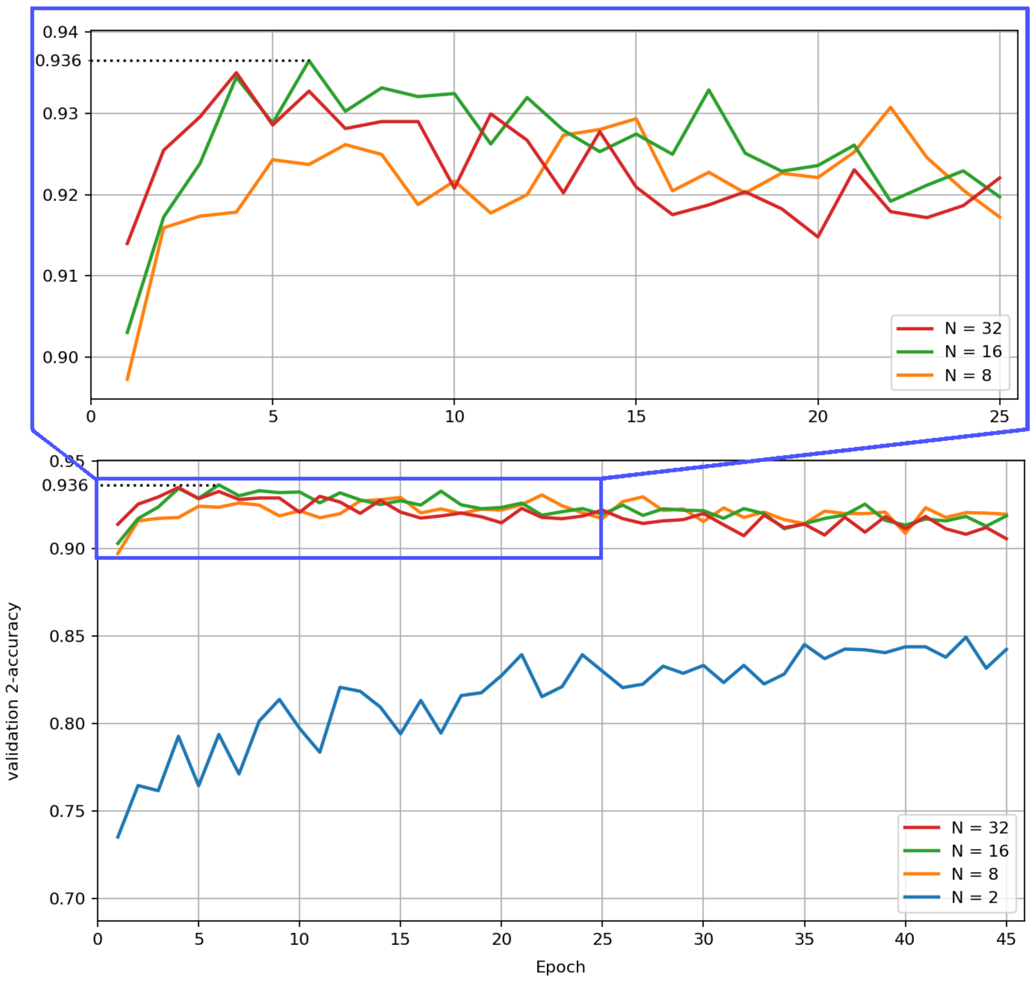


Figure 3.8 – *Validation 2-accuracy of SIGMOID-CC for different batch sizes.*

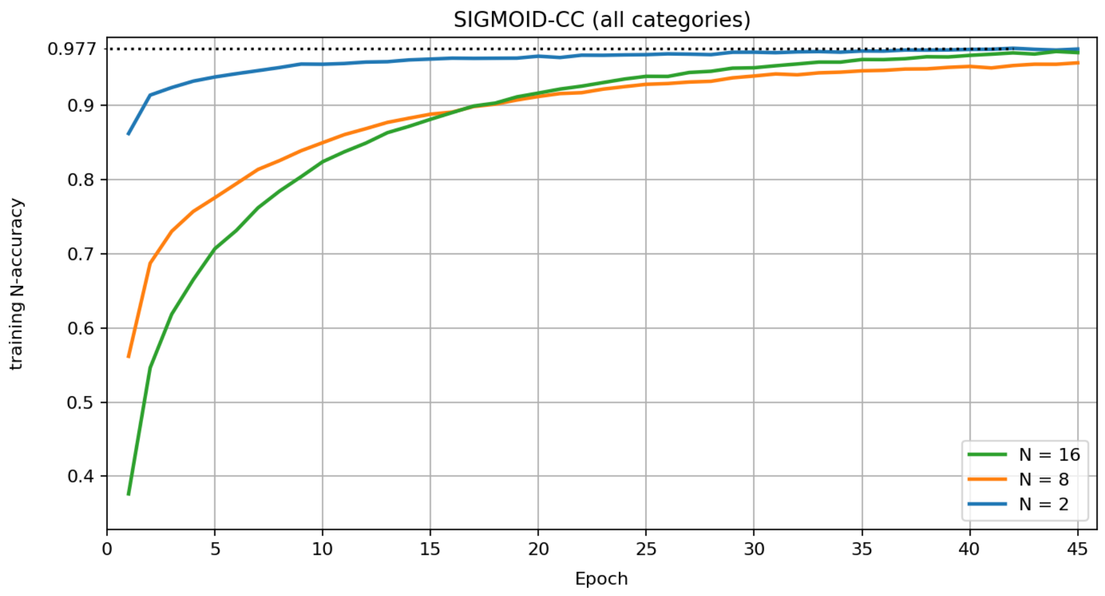


Figure 3.9 – *Training N-accuracy of SIGMOID-CC for different batch sizes.*

3.1.4 Sigmoid Loss Results

In order to compare the two losses, the very same training protocol has been used initially to conduct experiments varying N.

By looking at the 2-validation plot (fig. 3.8) it is clear that – as before – the 2^2 matrix does not provide an adequate level of generalization, despite an apparently perfect training (fig. 3.9).

The most interesting finding, however, is that the over-fitting we have previously described for the higher batch sizes it is now extremely moderate and does not seem to correlate with N. This is clearly an effect of Sigmoid loss, that allows to decouple the idea of contrastive training and batch size dependence.

Again, with the objective of achieving good generalization, the use of Sigmoid Loss seems superior when looking at the accuracy value, too. Fig. 3.10 shows a comparison of the results of 2-accuracy on validation data for the models trained with CLIP and SIGMOID (for both N=8 and N=16): Sigmoid Loss always makes the model perform better, especially for N=16.

Interestingly, this difference is not present in training accuracies (see fig. 3.11), that leads to the conclusion that the model is able to learn effectively with both losses, but Sigmoid favours generalization.

In fig. 3.7 it had been shown that, in the case of CLIP Loss, similar values of text and shape accuracies were not associated to good generalization, but rather to over-fitting. This is confirmed with Sigmoid Loss, since the two dimension-wise losses do not appear to correlate much more when N increases (with the possible exception of N=2, that exhibits more variability) and the model does not over-fit significantly.

One last important bit to evaluate training with Sigmoid is to check its performances on HST. If we look at how different N behave (fig. 3.12), we see that, as usual, N=2 does not perform well and that the best performances are obtained with N=16. Notably, further increasing N to 32 is not beneficial and even seems to slightly hinder performances.

To compare Sigmoid and CLIP, instead, we can look at the two best models in terms of validation accuracy (fig. 3.13): it is evident that Sigmoid generalizes better.

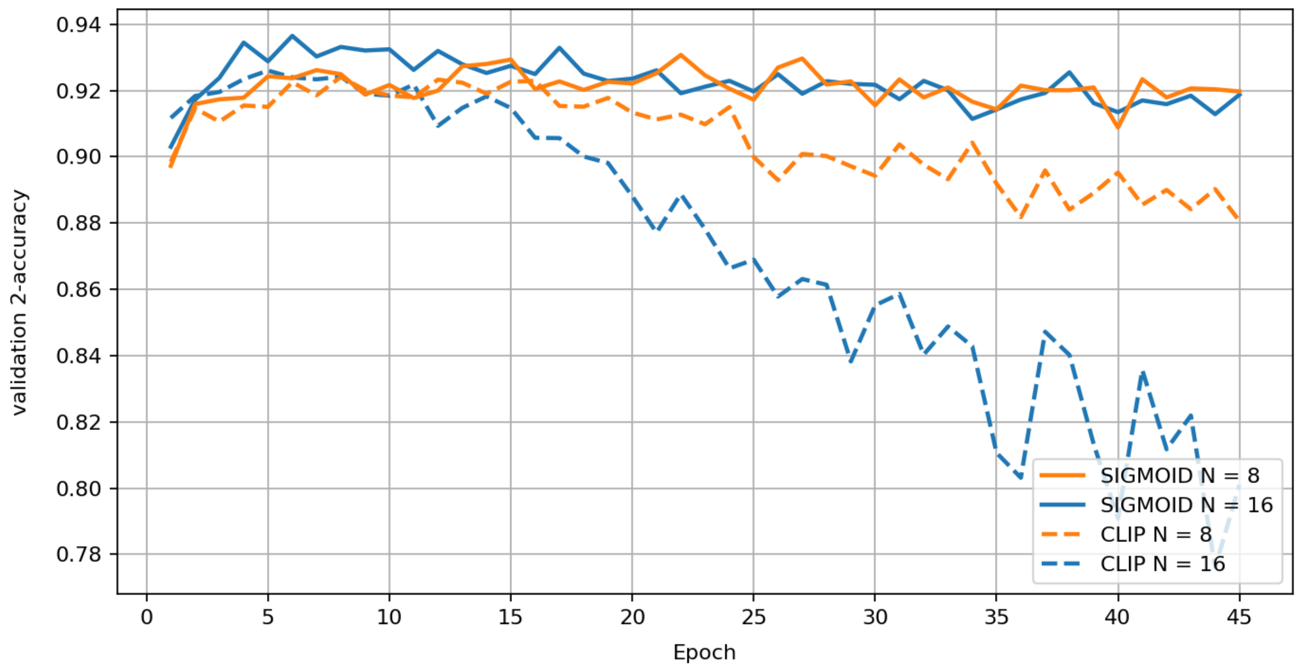


Figure 3.10 – *Comparison of validation 2-accuracies of SIGMOID-CC and CLIP-CC, for different batch sizes.*

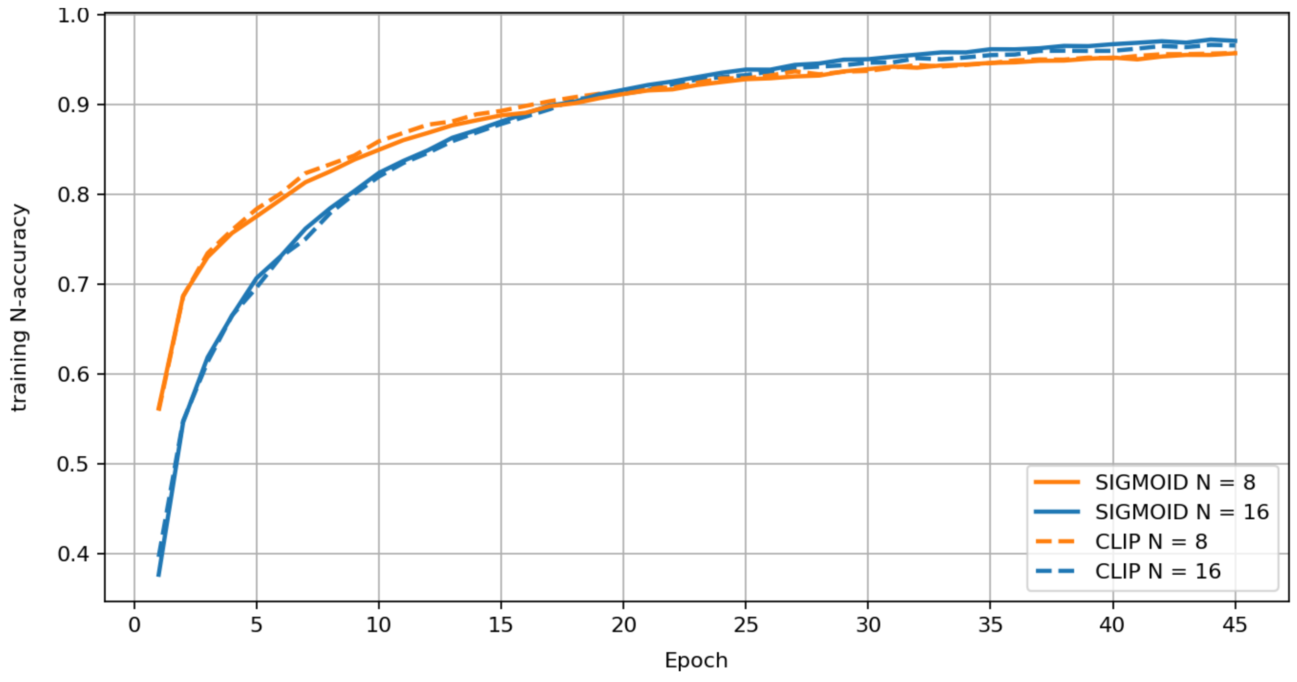


Figure 3.11 – *Comparison of training N-accuracies of SIGMOID-CC and CLIP-CC, for different batch sizes.*

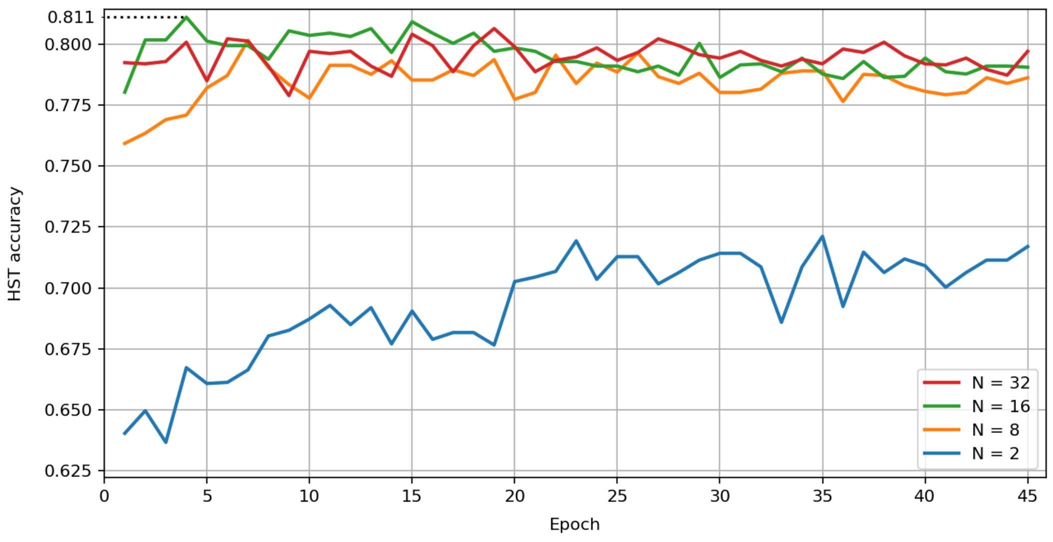


Figure 3.12 – *HST accuracy of SIGMOID-CC for different batch sizes.*

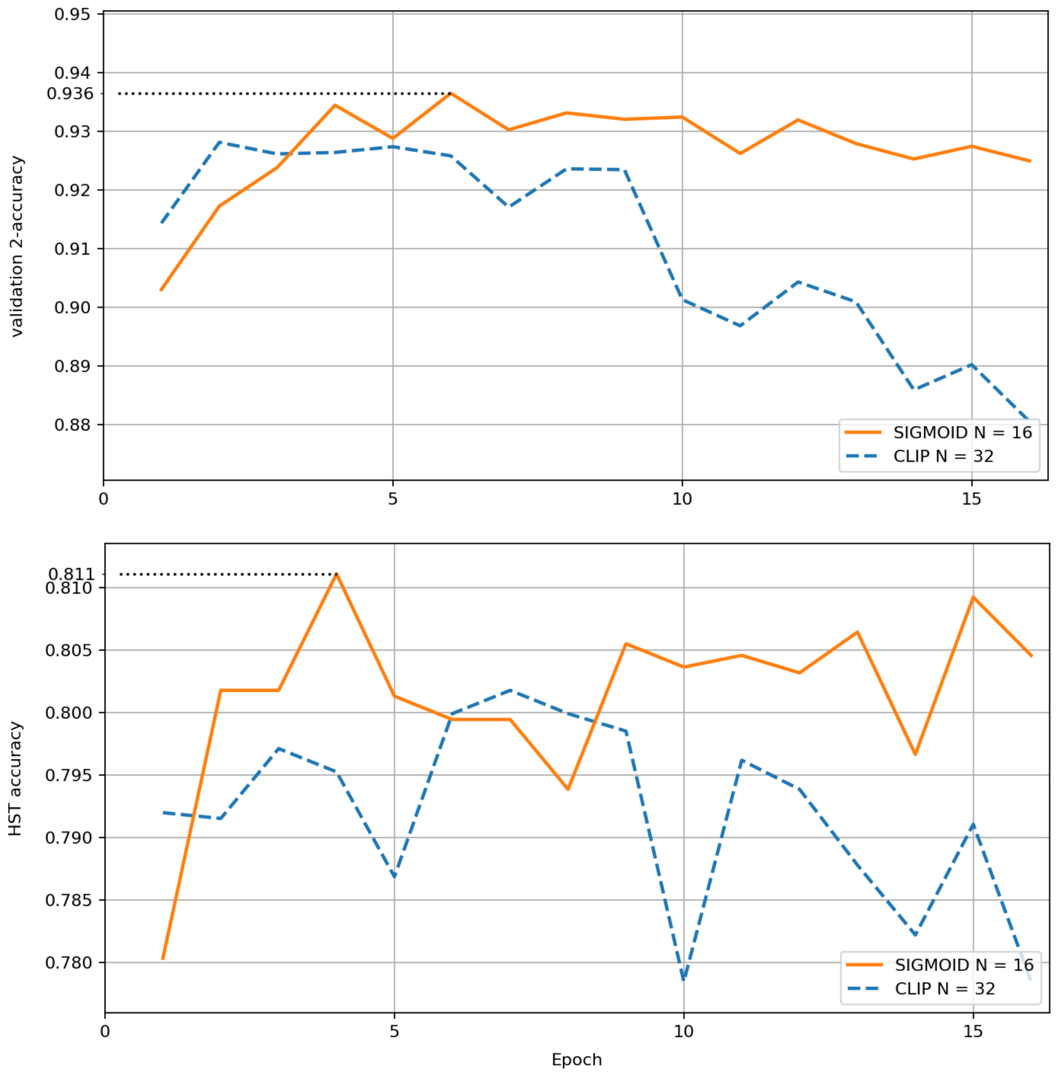


Figure 3.13 – *Comparison of validation 2-accuracy (top) and HST accuracy (bottom) of SIGMOID-CC and CLIP-CC for the batch sizes that obtained the best overall validation 2-accuracies.*

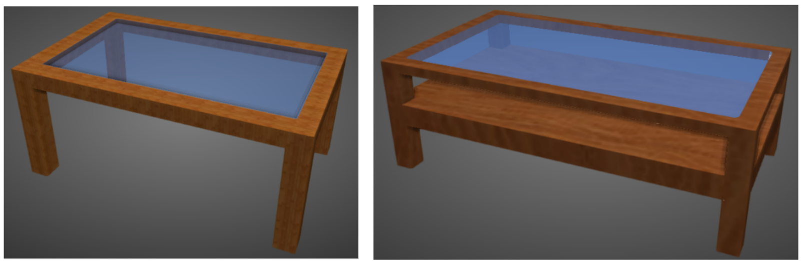
3.1.5 Comparison and Visual Results

We can finally summarize the best results we have so far on HST[4]:

Model/Loss	**HST accuracy**	**val. 2-accuracy**
CC, as reported by [Amaduzzi et al., 2023]	80.45%	–
CLIP-CC, N=8	80.3%	92.41%
CLIP-CC, N=32	80.18%	92.81%
SIGMOID-CC, N=16	**81.11%**	**93.65%**

It is important to highlight that, due to non-determinism in training, these data are not to be considered definitive. Uncertainty of measures should be estimated, for example by multiple repetitions of the experiments.

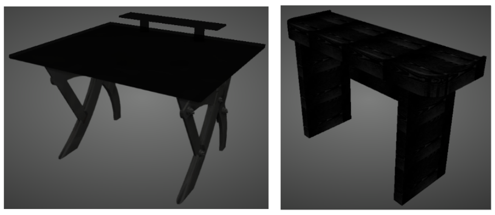
To get a better idea of the capability of the metric, when training the network with Sigmoid and CLIP-based losses, it is useful to visualize some of the errors it makes. Below, some errors of one or both kind of models are listed. Each example, taken from HST, includes a render of the two shapes, the textual description and the loss values, that are red when the model choose the corresponding shape wrongfully, green if the shape is chosen rightfully.



Rectangular dark brown table with glass top and sturdy wooden legs.

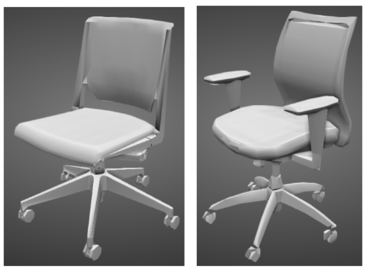
	left	right
CLIP-CC, N=32	**6.9149**	6.5262
SIGMOID-CC, N=16	**1.0226**	0.4379

[4]Note that, to avoid the distribution shift between HST and our validation dataset to hinder performances, HST data are treated here as validation data rather than test data: the best epoch is selected by looking directly to HST accuracy, rather than validation accuracy. A proper comparison will be presented later, using different training data.



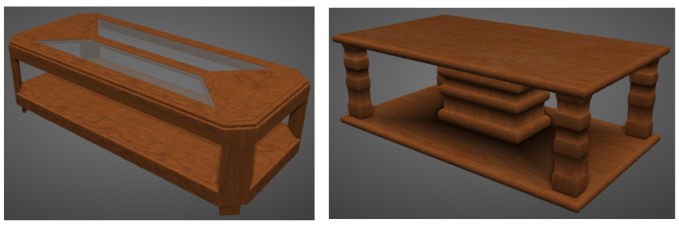
Rectangular black table with two steel legs, a wooden platform and a square base with a wooden rack in the middle.

	left	right
CLIP-CC, N=32	**4.3878**	1.7053
SIGMOID-CC, N=16	**0.0493**	-1.1318



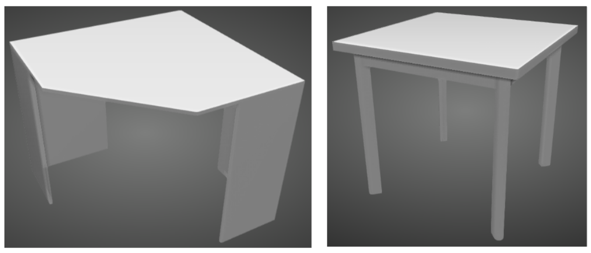
Sleek silver office chair with five wheels, two armrests, and an arched backrest.

	left	right
CLIP-CC, N=32	**4.1045**	3.7950
SIGMOID-CC, N=16	**1.0116**	3.5318



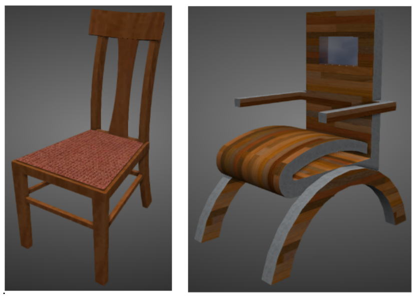
Rectangular wooden tea poy with four pillars, brown in color and carved details.

	left	right
CLIP-CC, N=32	1.0116	**3.5318**
SIGMOID-CC, N=16	**-0.1854**	-0.2216



a grey legged table with the white asymmetric top

	left	right
CLIP-CC, N=32	2.7660	**4.0073**
SIGMOID-CC, N=16	-0.5058	**0.2319**



Brown chair with 4 half-moon shaped legs, metal arm rests, and a glass window in the back.

	left	right
CLIP-CC, N=32	**2.0049**	1.5210
SIGMOID-CC, N=16	-1.1970	**-0.4456**

3.1.6 Analysing Single Categories and R-precision

To expand this analysis, it is useful to observe the models' behaviour when

1. training (and testing) inside a single category;
2. testing separately the two different distributions which we know HST is partitioned into;
3. testing under a protocol different than accuracy, such as the same used for CLIP R-precision.

To explore the first two aspects, only small adaptations of the code were necessary, given that both the category (*chair* or *table*) and the original set of captions (*GPT2shape* or *T2S*) are available as metadata of our dataset.

To test R-precision, a script used for the same purpose by [Amaduzzi et al., 2023] has been adapted to work with the current architecture.

The results of these experiments are summarized in the following table (note that this is another run of experiments with respect of the ones in previous paragraph, thus some data may be slightly different) and are referred to the epoch at which the validation 2-accuracy was maximum. This means that fine-tuning on HST distribution may further increase these accuracies.

	HST Accuracy				R-precision (full HST)
	Chairs only	GPT2shape only	T2S prompts only	Full	
CC, as reported	**81.04**	–	–	80.45	16.85
CLIP-CC, N=16	78.66	82.1	77.2	79.70	18.64
SIGMOID-CC, N=16	77.36	**83.8**	**78.7**	**81.33**	**19.73**
SIGMOID-CC, N=32	–	82.3	77.7	80.07	19.10

Interestingly, the original CC works better when dealing with shapes of a single category.

In every other case, training CC with Sigmoid Loss gives better performances. Particularly, the high R-precision values shown from contrastive techniques are significant and seems to support that contrastive losses, and especially Sigmoid, are good for retrieval tasks.

At last, it worth noting that – as expected – the highest accuracy values comes from that part of HST that is closer to training data distribution. Even if this may be explained by the less descriptiveness of plain HST w.r.t GPT2S, the fact that HST is human-validated should reduce the importance of this factor, in favour of the "(over)fitting to distribution" explanation.

3.2 Extension to Objaverse

3.2.1 Motivation and architectural adaptations

Give the interesting results obtained with CC on T2S, it worth exploring the behaviour of this architecture when provided with far more variability in its training data.

To achieve this, it is required a dataset with paired shape-description samples. This dataset has been identified with Cap3D [Luo et al., 2023], that pairs Objaverse shapes and relative textual descriptions, introduced in § 1.4.3.

When dealing with a larger pool of objects, one important problem arises: PointNet++, used as point encoder $\mathcal{E}_S$, has not been trained to handle shapes other than chairs and tables. At least two remedies exist: (a) re-train PointNet++ or (b) identify another architecture suitable to implement $\mathcal{E}_S$. The second possibility appears far more practical, due to the existence of architectures already trained on Objaverse shapes.

Particularly, PointLLM [Xu et al., 2023] provides a point cloud encoder based on Point-BERT [Yu et al., 2022], already trained on Objaverse shapes by a method dubbed ULIP-2, that [Xue et al., 2023] accept point clouds (instead of meshes).

To use this encoder, a few architectural modifications related to tensor dimensionalities are needed. Details are in fig. 3.14.

3.2.2 Training Experiments

Given that Sigmoid Loss has proven to be less prone to over-fitting in previous experiments, CC on Cap3D has been trained with that loss function.

Point-BERT encoder has been used frozen, as provided by [Xu, 2023; Xu et al., 2023]. Since only the point clouds used for training were available from the same source, those have been split in three parts (90%, 5%, 5%) to realize a train-validation-test split. Note that in this way, when testing the model,

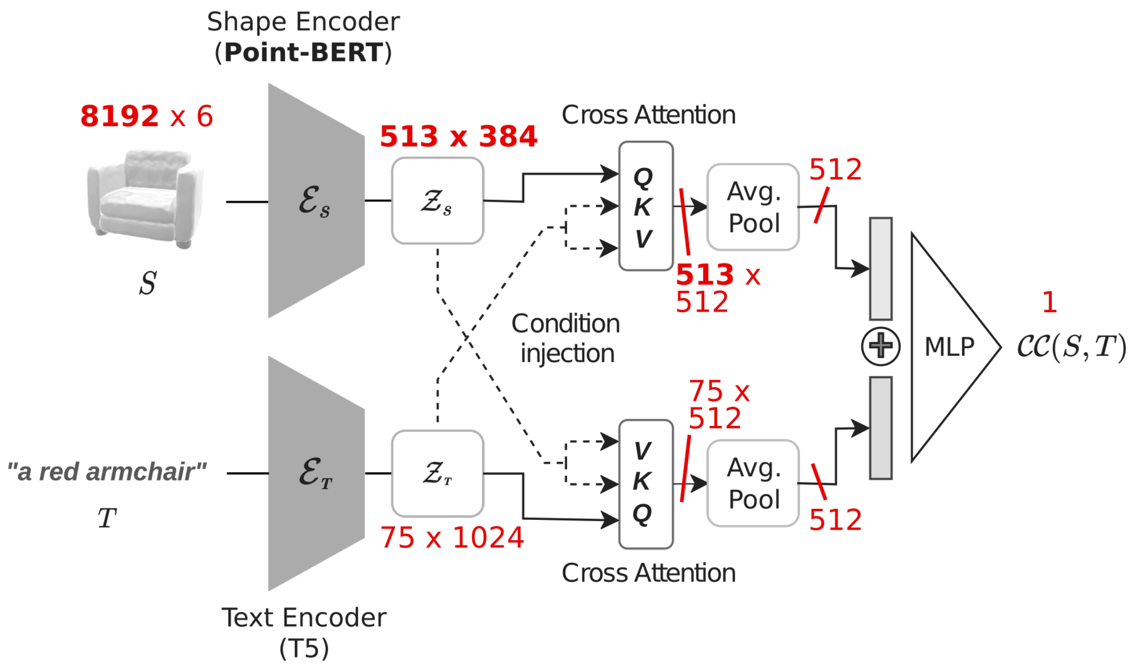


Figure 3.14 – *Data/activations dimensionality at each point in CC architecture (non-batched case), when using Objaverse clouds as input and Point-BERT as a point cloud encoder. The main differences from the previous architecture are bolded.*

the frozen encoder has already seen the clouds and therefore the test does not fully informs us about generalization performances to new data, but is only significant with respect to the performances of the remaining (unfrozen) part of the architecture.

Testing the Encoder

As a first step, the pre-trained encoder has been tested to check the significance of the produced features. Notably, the normalization function provided by the the authors in their code is not to be applied to training data (as it has been verified by the experiments summarized in the following fig. 3.15) which, however, neither appear to lie in the 3D sphere of radius 1, as it would be if the normalization had been already applied. This could make utilization of the model with new data problematic. For these preliminary experiments the provided (possibliy unnormalized) clouds been used *as is*, since this gives the higher semantic value to the encoder results among what have been tried.

As we can infer from fig. 3.15:

- similarity is significant both when max-pooling Point-BERT last layer activations and when just concatenating them. The latter option has been selected to train CC, to leave the model the possibility to better learn position-related information.

- Shape and colour are both taken into account by the encoder, at a degree that — at least for colours that are properly normalized in a fixed range — shows some dependence from input data magnitude.

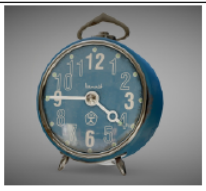
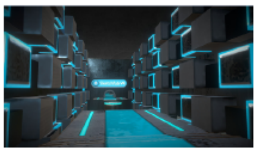
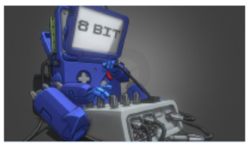
	norm, colors	norm, no colors	no norm, colors	no norm, no colors	no norm, colors	no norm, colors*0.5
	MAX_POOL				CAT	
	0.727	0.99148	0.432	0.463	0.5	-
	0.835	0.9992	**0.8**	0.856	0.825	-
	-	-	0.76	**0.912**	**0.84**	0.813
	0.658	0.98292	0.31	0.411	0.41	-
	0.754	**0.99943**	0.24	0.388	0.413	-
	0.706	0.99663	0.411	0.513	0.624	-
	0.89	0.99934	0.395	0.416	0.425	-
	0.841	0.98074	0.215	0.307	0.438	-
	0.758	0.98859	0.352	0.483	0.465	-

Figure 3.15 – *Cosine similarity values between the embedding of top image and the embeddings of the other images provided by pre-trained Point-BERT. Different configurations have been tested: with and without the normalization function provided by the authors; with the whole clouds (colours) and with zeroed colour values; max-pooling Point-BERT final activations or concatenating them.*

Training

Training has been conducted with the same protocol used for previous experiments. As for training on T2S, the model is able to reach high accuracy on training data; the only relevant difference is that convergence is way faster with Objaverse shapes, as expected with more data per epoch (compare fig. 3.16 and 3.9).

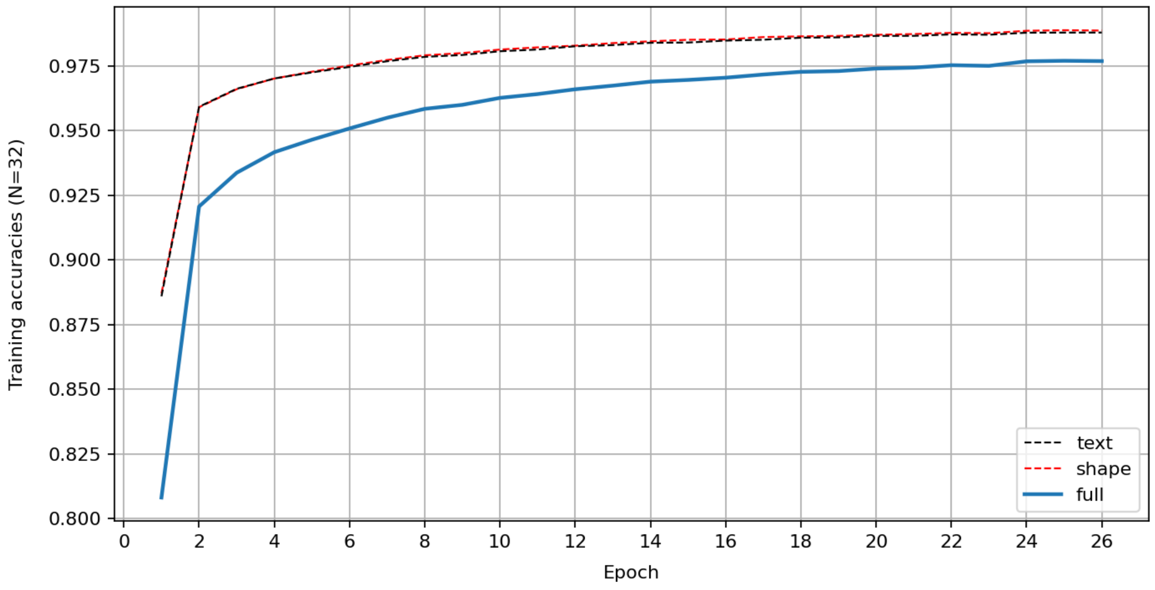


Figure 3.16 – *Text, shape and full training accuracies for N=32, when training on Cap3D/Objaverse data.*

It is more interesting that in Cap3D/Objaverse training validation accuracies are very high too, indicating good generalization capabilities. Particularly, 2-accuracy is too high to be informative (0.9983 at the best epoch), while 32-accuracy is displayed in fig. 3.17.

3.2.3 Building a Hard Dataset and Testing

When increasing data diversity, the model has to discriminate among very different shapes and texts. It is clear, than, that the task the model has to solve by contrastive learning becomes easier.

For this reason, after a test performed, as usual, by looking at the N-accuracy over the matrix built randomly, another approach has been tried. Specifically, a similar protocol to the one applied by [Amaduzzi et al., 2023] has been followed to test on a hard dataset.

In order to build a hard dataset it is possible to look at the distance among the features of the clouds encoded by Point-BERT, or by the texts encoded by T5. If the features are meaningful, an appropriate

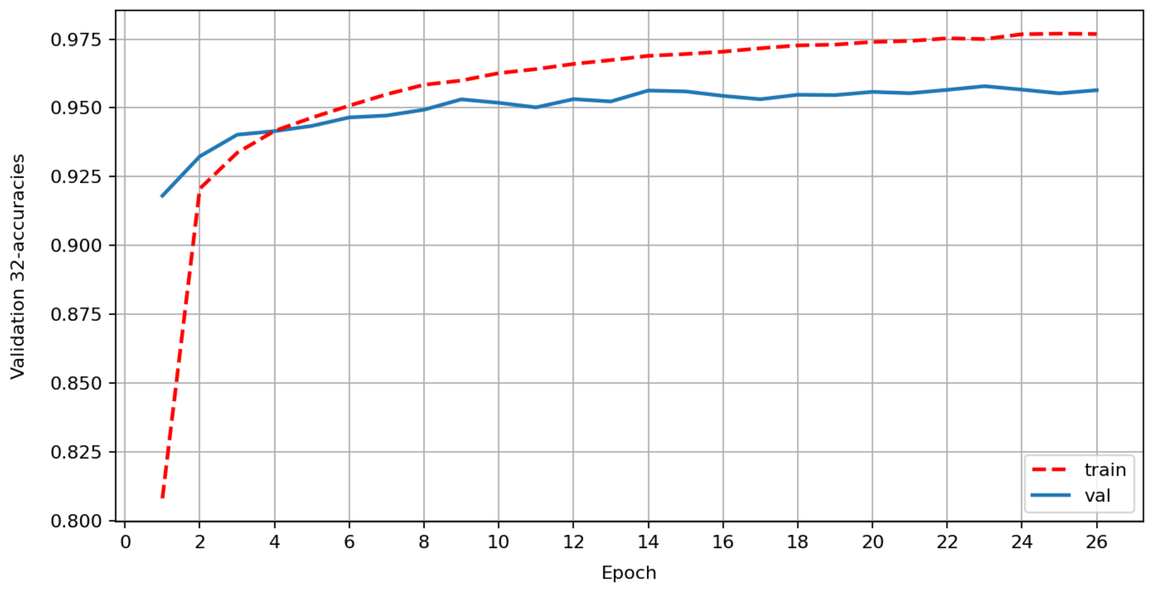


Figure 3.17 – *Validation 32-accuracies, when training on Cap3D/Objaverse data.*

distance-based query in that space would identify similar shapes or texts. Both paths were initially tried, realizing that, being Cap3D descriptions much less informative than Objaverse point cloud, vicinity in Point-BERT embedding space is much more significant and allows to create a good *hard* dataset.

Particularly, to ease implementation process, k-NN in the embedding space has been used to identify at least a neighbourhood for each encoded shape. Three different hard datasets were created, sampling the distractor shape from a different set of neighbourhood:

- **K=1**, that associates each shape with the one with closest embedding;
- **K=25**, that associates each shape with one uniformly sampled from the 25 shapes with closest embedding;
- **K=50**, that associates each shape with one uniformly sampled from the 50 shapes with closest embedding.

Figure 3.18 illustrate the different kind of associations that exist in this datasets.

The protocol used (see [Amaduzzi et al., 2023, § 7.1]) consists in computing CC between a text t_1 and two shapes s_1 and s_2 separately, to get a value for (t_1, s_1) and another for (t_1, s_2) and report the fraction of times that the model computes the highest coherence for the ground-truth shape. Test results are

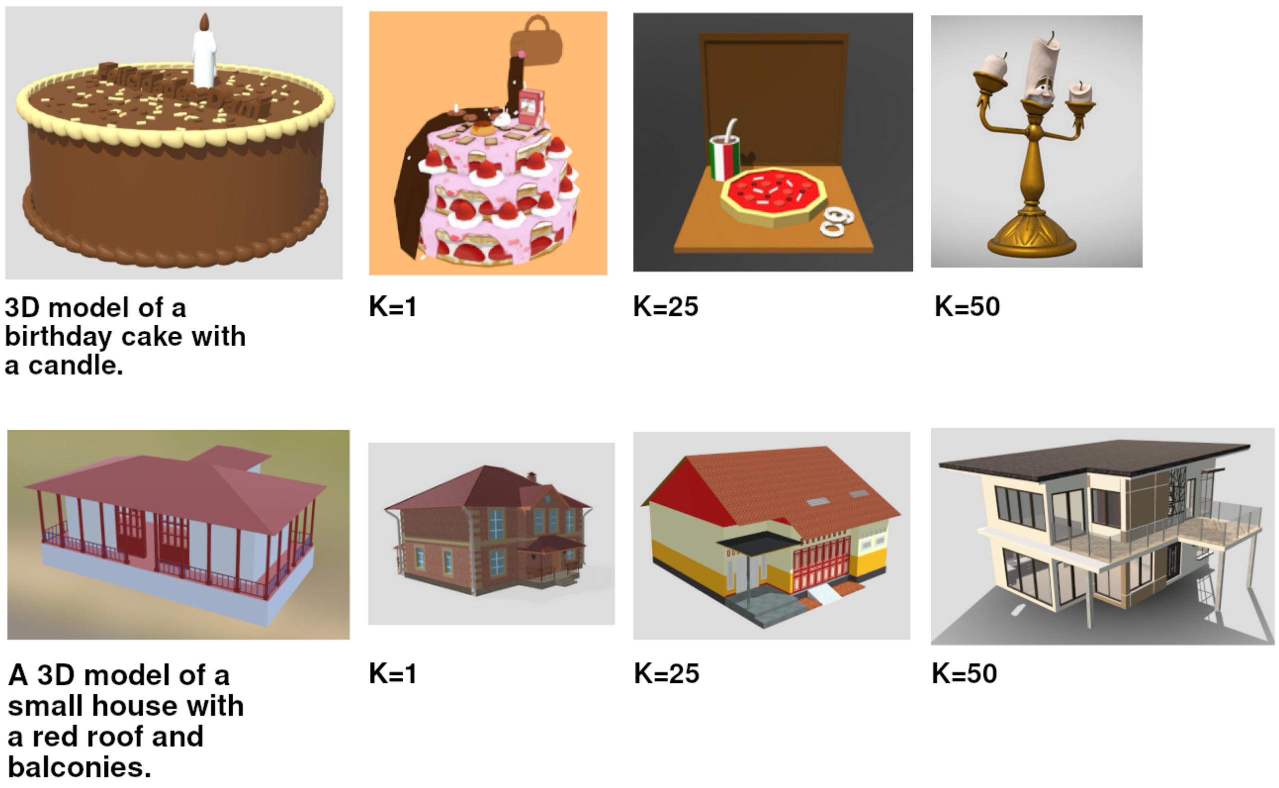

3D model of a birthday cake with a candle.
K=1
K=25
K=50
A 3D model of a small house with a red roof and balconies.
K=1
K=25
K=50

Figure 3.18 – *Samples taken from the hard datasets built with K-NN in Point-BERT embedding space.*

the following.

	K = 1	K = 25	K = 50
Accuracy	78.54%	89.47%	92.21%

It is important to remember that these results only indicates how good the model is to associate text and shapes, when excluding the encoders part of the architecture, as anticipated in § 3.2.2. A useful extension of this work would include a testing procedure with entirely new data.

Finally, the text-shape correspondence has been analysed by looking at the error that the model makes, confirming that a relevant part of them are due to ambiguous description-shape correspondence, especially for distant values of cross coherence computed by the model. One error example is reported 3.19.

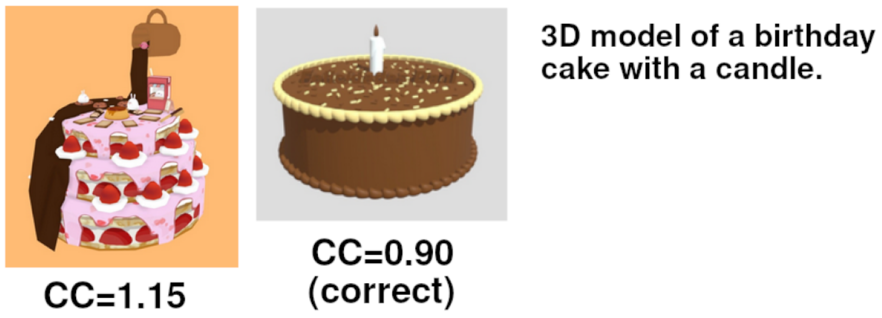


Figure 3.19 – *Error of the model trained on Objaverse/Cap3D.*

CONCLUSION

In this work the interplay of two important factors for performances of a multi-modal network has been explored: contrastive training, with a particular focus on the loss function, and the scale and variability that characterize the training dataset.

During this exploration, the aforementioned factors have been explored on a recently proposed metric, CrossCoherence.

Contrastive losses have confirmed their effectiveness, although sensitive to the risk of over-fitting (CLIP loss) and to batch size (both). Furthermore, contrastive training has shown to be even more effective when paired with a large-scale dataset. In this scenario, CrossCoherence has proven the efficacy of its design.

Still, some ideas were left unexplored, such as CrossCoherence behaviour with batch sizes higher than 32 (due to lack of hardware resources and time) and with brand new data (in the case of CC trained on Objaverse/Cap3D), in addition to others that can be found in the text.

Finally, **some people I would like to thank**.

My supervisor, Samuele Salti, that introduced me to computer vision with clarity and rigorous passion and Andrea Amaduzzi, without whose willingness and wise guidance this thesis would not exist (not yet, at least!). My parents, whose example made me who I am and my brother, who loves me in his own way. Doroty, who with her love supported me in every moment of the last two years. And, finally, my students in Cento, that taught me that nothing can be taken for granted.

BIBLIOGRAPHY

Articles

Amaduzzi, A., Lisanti, G., Salti, S., and Di Stefano, L. (2023). “Looking at words and points with attention: a benchmark for text-to-shape coherence”. In: *2023 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)* (cit. on pp. 5, 13, 14, 19, 28, 40, 43, 48, 49).

Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., and Yu, F. (2015). *ShapeNet: An Information-Rich 3D Model Repository* (cit. on p. 13).

Chen, K., Choy, C. B., Savva, M., Chang, A. X., Funkhouser, T., and Savarese, S. (2018). *Text2Shape: Generating Shapes from Natural Language by Learning Joint Embeddings* (cit. on pp. 11–13, 17).

Deitke, M., Schwenk, D., Salvador, J., Weihs, L., Michel, O., VanderBilt, E., Schmidt, L., Ehsani, K., Kembhavi, A., and Farhadi, A. (2022). *Objaverse: A Universe of Annotated 3D Objects* (cit. on p. 15).

Fu, R., Zhan, X., Chen, Y., Ritchie, D., and Sridhar, S. (2023). *ShapeCrafter: A Recursive Text-Conditioned 3D Shape Generation Model* (cit. on pp. 12, 13, 18).

Gao, K., Gao, Y., He, H., Lu, D., Xu, L., and Li, J. (2023). *NeRF: Neural Radiance Field in 3D Vision, A Comprehensive Review* (cit. on p. 7).

He, Y., Bai, Y., Lin, M., Zhao, W., Hu, Y., Sheng, J., Yi, R., Li, J., and Liu, Y.-J. (2023). *T^3Bench: Benchmarking Current Progress in Text-to-3D Generation* (cit. on p. 22).

Jun, H. and Nichol, A. (2023). *Shap-E: Generating Conditional 3D Implicit Functions* (cit. on p. 12).

Li, J., Li, D., Savarese, S., and Hoi, S. (2023). *BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models* (cit. on p. 16).

Lipman, Y., Sorkine, O., Levin, D., and Cohen-Or, D. (2005). “Linear rotation-invariant coordinates for meshes”. *ACM Trans. Graph.*, 24(3), pp. 479–487. URL: `https://doi.org/10.1145/1073204.1073217` (cit. on p. 10).

Liu, Z., Wang, Y., Qi, X., and Fu, C.-W. (2022). *Towards Implicit Text-Guided 3D Shape Generation* (cit. on pp. 12, 17).

Luo, T., Rockwell, C., Lee, H., and Johnson, J. (2023). *Scalable 3D Captioning with Pretrained Models* (cit. on pp. 16, 44).

Meagher, D. (1980). *Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer.* Tech. rep. IPL, Rensselaer Polytechnic Institute, New York (cit. on p. 8).

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. (2020). *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis* (cit. on p. 7).

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation* (cit. on pp. 21, 22).

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. (2021). *Learning Transferable Visual Models From Natural Language Supervision* (cit. on pp. 17, 27).

Sanghi, A., Chu, H., Lambourne, J. G., Wang, Y., Cheng, C.-Y., Fumero, M., and Malekshan, K. R. (2022). *CLIP-Forge: Towards Zero-Shot Text-to-Shape Generation* (cit. on p. 17).

Tatarchenko, M., Dosovitskiy, A., and Brox, T. (2017). “Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs”. *CoRR*, abs/1703.09438. URL: `http://arxiv.org/abs/1703.09438` (cit. on p. 8).

Xu, R., Wang, X., Wang, T., Chen, Y., Pang, J., and Lin, D. (2023). *PointLLM: Empowering Large Language Models to Understand Point Clouds.* URL: `https://github.com/OpenRobotLab/PointLLM` (cit. on p. 44).

Xue, L., Yu, N., Zhang, S., Li, J., Martín-Martín, R., Wu, J., Xiong, C., Xu, R., Niebles, J. C., and Savarese, S. (2023). *ULIP-2: Towards Scalable Multimodal Pre-training for 3D Understanding* (cit. on p. 44).

Yu, X., Tang, L., Rao, Y., Huang, T., Zhou, J., and Lu, J. (2022). *Point-BERT: Pre-training 3D Point Cloud Transformers with Masked Point Modeling* (cit. on p. 44).

Zhai, X., Mustafa, B., Kolesnikov, A., and Beyer, L. (2023). *Sigmoid Loss for Language Image Pre-Training* (cit. on p. 34).

Zhu, X., Sun, P., Wang, C., Liu, J., Li, Z., Xiao, Y., and Huang, J. (2023). *A Contrastive Compositional Benchmark for Text-to-Image Synthesis: A Study with Unified Text-to-Image Fidelity Metrics* (cit. on p. 19).

Web Resources

Tiange (2023). *Cap3D (Scalable 3D Captioning with Pretrained Models)*. URL: `https://huggingface.co/datasets/tiange/Cap3D` (cit. on p. 16).

Xu, R. (2023). *Xu's HuggingFace repository*. URL: `https://huggingface.co/RunsenXu` (cit. on p. 44).