Combinatorial Generalization - Does learning Group Structured Representation really help?

Simone Caldarella
Artificial Intelligence Systems, DISI
University of Trento
Povo, TN 38123
simone.caldarella@studenti.unitn.it

Abstract

Deep neural networks are known for their ability to learn complex, high-dimensional representations of input data. These representations, however, often lack an explicit structure that can be easily interpreted by humans. Some recent machine learning approaches have attempted to address this issue by training neural networks to learn representations that are composed of "disentangled" elements, however those model still struggle with the challenge of combinatorial generalization, which involves understanding how disentangled properties combine in novel ways to create new objects or scenarios. To overcome this challenge, another class of machine learning approaches has been proposed, which aims to impose additional group structure on the learned representations, which improves the network's ability to generalize to new combinations. The purpose of this project is to investigate whether the use of group structure constraints can indeed improve combinatorial generalization in neural networks. Specifically, the model proposed by by Keurti et al. (2), i.e. Homomorphism AutoEncoder, is trained and tested following combinatorial generalization setup proposed by Montero et al. in (7) and (6), to guarantee a fair evaluation of model's combinatorial performance.

1 Introduction

Comprehending and creating new combinations of familiar elements, known as *Combinatorial Generalization*, is believed to be a fundamental ability of the human mind, posing a significant hurdle for neural network models that aims to mimic human behaviour. Numerous studies indicate that classical neural networks (e.g. autoencoders) are incapable of solving this problem unless they incorporate mechanisms tailored for symbol representation. Recent studies suggest that learning a disentangled representations of data may help address this problem enforcing the model to understand the different components of object seen during training. A famous work going in that direction is β-VAE (1), proposed by Higgins et al. in 2017, that aim to discover interpretable factorised latent representations (in image data) in a completely unsupervised manner.

Despite the enthusiasm on this approach to build compositional-aware generative models, the research accomplished in 2021 by Montero et al. (7) showed the lacks of this line of studies. The work not only formalizes the problem of Combinatorial Generalization but, to avoid unfair evaluations processes on combinatorial-tasks, proposes a three difficulty-levels benchmark to evaluate Combinatorial Generalization capabilities of given models.

Specifically, in order of increasing complexity (with examples related to latent factors of dSprite dataset (5)):

- Recombination-to-Element

```
[shape = ellipse, scale = 1, orientation < 120o,
    position-x > 0.5, position-y > 0.5]
```

- Recombination-to-Range

```
[shape = ellipse, scale = [all], orientation = [all],
    position-x > 0.5, posiion-y = [all]]
```

- Extrapolation

```
[shape = [all], scale = [all], orientation = [all],
    position-x > 0.5, posiion-y = [all]]
```

Each benchmarks represents the factors of the samples in the dataset that are left out during the training and that will be evaluated. Employing this leaving-out methodology makes possible to fairly assess how the model can reconstruct new examples without having seen similar ones (in a decreasing levels) during training. The results of this work, e.g. the benchmarks on previous SoTA compositional model, showed that learning a disentanglement representation was not a sufficient conditions to accomplish combinatorial generalization.

Recently, other works proposed better disentanglement representation learning strategies and amidst these, Keurti et al. proposed a model called Homomorphism Autoencoder (2), that takes in account the concept of transformation between combinations/samples to enforce the relations between composition of latent factors. Since the goal of learning disentanglment representation of the outside world, the authors starts their research with the following questions:

1. In what manner can we obtain world models that accurately depict the external environment both in terms of its actual state and the influence of our actions on it?
2. Is it feasible to acquire these models through interaction with the world, and can we establish mathematical criteria for their correlation with a hypothetical reality that exists beyond our cognition?

Unifying the questions leads to their proposed solution where an autoencoder equipped with a group representation that linearly acts on its latent space and is trained on a 2-step reconstruction, which enforces a homomorphism property on the group representation (the transitions/transformations). The authors motivate their method theoretically and demonstrate in the paper its empirical effectiveness in learning the correct representation of the groups and topology of the environment.

The driving aim of this project is to assess to which extents the Homomorphism Autoencoder can overcome the Combinatorial Generalization limitations, of the previous generative model, elicited by Montero et al. in "The role of Disentanglement in Generalisation" (7).

In the following sections we will discuss about:

- how dSprite dataset is built and how the Homomorphism Autoencoder code is structured, in order to understand how the tests have been performed 3
- the actual methodology applied to test the model on the combinatorial benchmark along with the changes in the HomomorphismAE code 4
- the experiment's details (e.g. parameters for the tests) and the results obtained.

2 Theoretical Motivations

In (7), Montero et al. show weaknesses of three variational autoencoders based architectures, i.e. Vanilla VAE, β-VAE and Factor VAE, in the domain of combinatorial generalization. The underlying criticism over these models is backed on the fact that learning solely a disentanglement representation of the data is not a sufficient condition to gain combinatorial generation capabilities. For the these reason, our investigation on combinatorial abilities of Homomorphism Autoencoder ((2)) originates from the different approach in which disentanglement learning is accomplished. Specifically, our research question is wether enforcing disentanglement learning by learning transitions between datapoints can actually be beneficial in a combinatorial context.

Before proceeding with the next section, let's go through the peculiarities of the previously cited variational autoencoder architectures in order to better understand how they differ from each other and from homomorphism autoencoder.

2.1 Vanilla Variational Autoencoder

Variational Autoencoders (VAEs) have been widely used for unsupervised learning and generative modeling. What characterizes the standard VAEs (4) is their probabilistic nature, since the input is mapped into a multivariate distribution instead of a fixed vector, as it happens with autoencoders. This allows to theoretically encode the generative factors in the dimensions of the latent distribution. However, since the reconstruction remains the primary task at training time, a trade-off between reconstruction fidelity (the first term) and latent space regularization (the second term with the Kullback Leibler divergence) is accomplished through its obejctive function:

$$\mathcal{L}(\theta, \phi; \mathbf{x}) = -\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] + \mathcal{D}_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$$

Despite the trade-off introduction, VAEs empirically learns a limited disentanglement of true generative factors, demaning for improved model to improve the learned representation.

2.2 β-VAE

β-VAE (1) introduces an additional parameter, denoted as β, whose aim is to vary the weight of the KL divergence term of VAEs objective. The purpose of this additional term is to explicitly encourage the learning of disentangled and interpretable representations in the latent space. By adjusting this parameter, it is possible to control the trade-off between reconstruction accuracy and the level of disentanglement in the learned representations. For example, when $\beta = 1$, it operates similarly to a standard Variational Autoencoder (VAE). However, when the $\beta > 1$, the objective applies a stronger constraint on the latent bottleneck, limiting the representation capacity. In cases where the generative factors are conditionally independent, maintaining disentangled representations becomes more effective. Therefore, a higher value of beta promotes more efficient latent encoding and further encourages disentanglement. However, it's important to notice that increasing beta may introduce a trade-off between reconstruction quality and the degree of disentanglement, increasing artifacts in the output (e.g. blurriness).

2.3 Factor VAE

Introduced in 2017, Factor VAEs (3) represent a step forward in learning disentanglement factors. The main weakness of β-VAEs is that penalizing the entire KL term in order to improve mapping-to-latent efficiency come at the cost of degraded reconstruction performance. For this reason Factor VAEs make a different use of the KL term. This term of the loss is indeed factorised as the sum of the mutual information between input and latent and the actual KL divergence between the latent and the prior:

$$\mathbb{E}_{p_{data(x)}}(KL(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))) = I(\mathbf{x}; \mathbf{z}) + KL(q(\mathbf{z})||p(\mathbf{z}))$$

Factorising the KL term allows to introduce a more balanced weighting in the objective function, since penalising the mutual information (I) is not necessary nor desirable for disentangling, while penalising the KL between latent and prior encourages independence in the dimensions of z, and thus disentanglment.

2.4 Homomorphism Autoencoder

Homomorphism Autoencoder ((2)) is a change of paradigm in dealing with disentanglement learning.

The choosen backbone is a standard autoencoder, thus probabilistic properties of the latents are not directly taken in account while designing the objective. The model aims to jointly learn a group representation of transitions between world states (the input images), as well as a symmetry based disentangled representation of observations (block diagonal matrices that encode transitions), with minimal assumptions on the group or the actions the agent can sample from.

Let's break out the above sentence to better understand the contribution of this new architecture. The high level idea is that by constraining the training to work in the space of images that are transformed through block diagonal matrices we can learn transitions that implicitly encode the disentangled nature of the latent factors.

The novelty of the approach, which theoretically achieves the same goal of previously cited VAEs, is what prompts us to inquire whether this setup brings benefits not only in learning disentangling factors but also in the domain of combinatorial generalization. Moreover the multi-step reconstruction, compared to the single step one performed in VAEs, could better enforce the *geometric effect* of a transformation (and thus of a generative factor that generate it) because of its consequential nature (step 2 is directly dependant on the step 1, which is again dependant on the step 0).

3 Implementation Background

3.1 Dataset: dSprites - Disentanglement testing Sprites dataset

The dSprites dataset (5), is a valuable resource for evaluating the disentanglement properties of unsupervised models. The dataset comprises 737,280 2D shape images generated through a procedural method involving six independent latent factors: color, shape, scale, rotation, and the x and y positions of a sprite. To ensure comprehensive coverage, all possible combinations of these latents are precisely represented in the dataset. The latent factor values were thoughtfully selected to provide minimal step changes while guaranteeing distinct pixel outputs. Notably, no additional noise was introduced. The dataset is stored in a NPZ NumPy archive and is also available in a HDF5 version and it can be accessed through the repository of the project `https://github.com/deepmind/dsprites-dataset`. This dataset, originally developed for the evaluation of the β-VAE architecture (1) serves as a rigorous benchmark for assessing the disentanglement quality of unsupervised models. It allows researchers to evaluate how effectively models recover the ground truth latents.

From the metadata provided along with the dataset is possible to extract all the available combinations given by the following values:

```
metadata = {
    'latents_names': ('color', 'shape', 'scale',
                      'orientation', 'posX', 'posY'),
    'latents_possible_values': {
        'orientation': array([0.        , 0.16110732, 0.32221463, 0.48332195,
                              0.64442926, 0.80553658, 0.96664389, 1.12775121,
                              1.28885852, 1.44996584, 1.61107316, 1.77218047,
                              1.93328779, 2.0943951 , 2.25550242, 2.41660973,
                              2.57771705, 2.73882436, 2.89993168, 3.061039  ,
                              3.22214631, 3.38325363, 3.54436094, 3.70546826,
                              3.86657557, 4.02768289, 4.1887902 , 4.34989752,
                              4.51100484, 4.67211215, 4.83321947, 4.99432678,
                              5.1554341 , 5.31654141, 5.47764873, 5.63875604,
                              5.79986336, 5.96097068, 6.12207799, 6.28318531]),

        'posX':        array([0.        , 0.03225806, 0.06451613, 0.09677419,
                              0.12903226, 0.16129032, 0.19354839, 0.22580645,
                              0.25806452, 0.29032258, 0.32258065, 0.35483871,
                              0.38709677, 0.41935484, 0.4516129 , 0.48387097,
                              0.51612903, 0.5483871 , 0.58064516, 0.61290323,
                              0.64516129, 0.67741935, 0.70967742, 0.74193548,
                              0.77419355, 0.80645161, 0.83870968, 0.87096774,
                              0.90322581, 0.93548387, 0.96774194, 1.        ]),

        'posY':        array([0.        , 0.03225806, 0.06451613, 0.09677419,
                              0.12903226, 0.16129032, 0.19354839, 0.22580645,
```

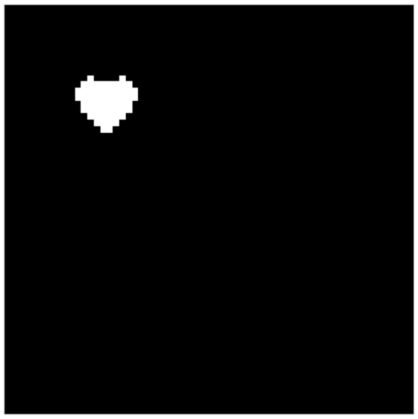

Figure 1: Visualization of a single heart.

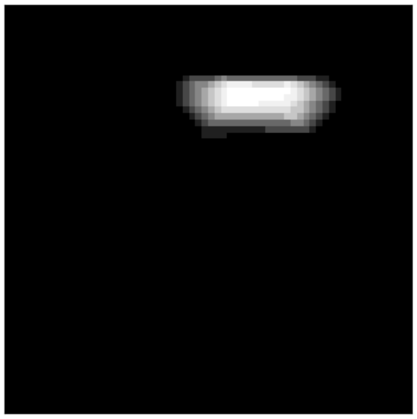

Figure 2: Visualization of range of hearts.

```
                                    0.25806452, 0.29032258, 0.32258065, 0.35483871,
                                    0.38709677, 0.41935484, 0.4516129 , 0.48387097,
                                    0.51612903, 0.5483871 , 0.58064516, 0.61290323,
                                    0.64516129, 0.67741935, 0.70967742, 0.74193548,
                                    0.77419355, 0.80645161, 0.83870968, 0.87096774,
                                    0.90322581, 0.93548387, 0.96774194, 1.        ]),
     'scale': array([0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
     'shape': array([1., 2., 3.]),
     'color': array([1.])},
 'latents_sizes': array([ 1,  3,  6, 40, 32, 32])}
```

In order to simplify the sampling process we will refer to the index of the factor's value instead of the value itself (e.g. posX = 1 means posX = 0.03225806). For example, given the following properties:

```
[shape = 2, posX = 0, posY = 0, orientation = 0, scale = 0]
```

we will obtain the heart in fig. 1. The generative factor *color* is not taken in account due to its univoque value.

If more images are sampled, we can visualize them all at once using a simplified concept of image density, whose definition is contained in dSprite repository tutorial `https://github.com/deepmind/dsprites-dataset/blob/master/dsprites_reloading_example.ipynb`. For instance we can extract the following range of images:

```
[shape = 2, posX >= 15, posY = 0, orientation = 0, scale = 0]
```

and check the result in fig. 2.

3.2 Model: Homomorphism Autoencoder

Grasping how Homomorphism Autoencoder is implemented is crucial to understand how to inject our changings without modifying the beahviour of the model itself, ensuring a fair evaluation of the architecture. The code is accessible from the repository of the project at `https://github.com/hamzakeurti/homomorphismvae`.

The pipeline in the code involves the following steps:

- in the $\_\_init\_\_()$ of $DspritesDataset()$:
  - all indices related to the choosen parameters are mapped into a continuous list
  - with

    `self.train_start_idx =`

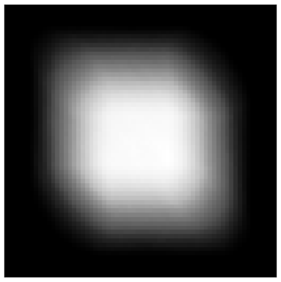
Figure 3: Density of training images.

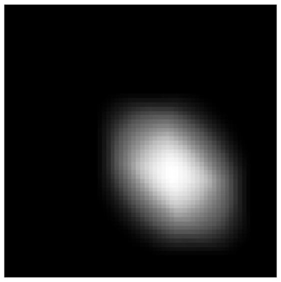
Figure 4: Density of recomb2range test images.

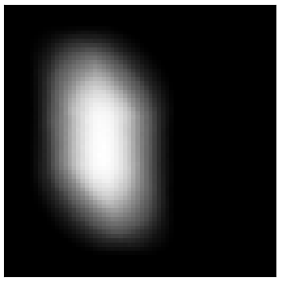
Figure 5: Density of training images without test ones.

```
self._rand.choice(self.dataset_size, size=num_train)
```

a list of num_train idxs are choosen

- with

```
self.train_idx, self.train_dj =
    self.observe_n_transitions(self.train_start_idx)
```

two lists are returned:

  * $self.train\_idx$ is a list of triplets in the format $[[idx1\_1, idx1\_2, idx1\_3]$, ..., $[idxN\_1, idxN\_2, idxN\_3]]$, where each triplet contains the three images indices whose transition are computed
  * $self.train_dj$ is a list of couples in the format $[[dj1\_12, dj1\_23]$, ..., $[djN\_12, djN\_23]]$, where each dj is a 2×2 matrix which represent the transition from one image to another

- in the $\_\_getitem\_\_()$ of $DspritesDataset()$:
  - the batch of idxs triplets and dj couples are returned

By dissecting the code it is possible to extract the list of all images that are used and check their distribution in fig. 3.

```
imgs, metadata = load_dsprite()
show_density(imgs[dhandler.all_indices[dhandler.train_idx.flatten()]])
```

This analysis allows to better understand the impact of removing a subset of the images used by the model.

Specifically, considering the following range of images for a *Recombination-to-Element* benchmark:

```
[shape = 1, orientations = 14, scale = 5, posX >= 15, posY >= 15)
```

we can se in the pictures from fig. 3 to fig. 8 the effect on removing the image range from training (fig. 5 and from test (fig. 8.

An important limitation that it needs to be kept in account is related to the actual combinations considered by the model. With the proposed model, only the posX and posY generative factors combinations are considered and the others are kept fixed at `[scale=5, orientation=14, shape=1]`.

They also proposed a model with all the shapes but as for now we will consider only the procedure made only on ellipses (shape=1).

From now on both the basics concept related to the task and the implementations details should be coarsly understood.

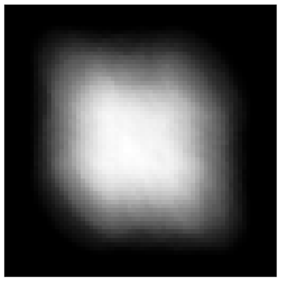
Figure 6: Density of test images.

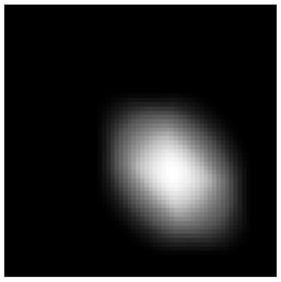
Figure 7: Density of recomb2range test images.

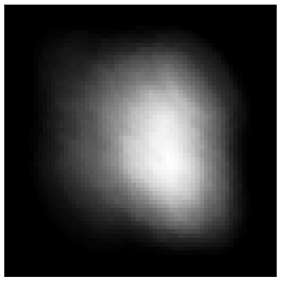
Figure 8: Density of test images without test ones.

4 Method

4.1 Extracting Benchmark images

Starting from the dSprites dataset, one may want to select a set of latent factors and obtain the set of images that are combinations of the requested factors. To do so, the method $exclude_from_training$ (code implementation in appendix A) allows to select the indices of the factors requested and returns the list of images containing the combination of these latents. For example, to get the entire set of images that have $posX > 0.32221463$ (index > 2) it is possible to write:

```
exclude_from_training(imgs, metadata, posX="> 2")
    -> [imgs, idxs, combination_name]
```

where *imgs* and *metadata* are respectively the set of images and the metadata dict shown before and where all the non specified factor values are considered as the entire range of values.

4.2 Modifications in HomomorphismAE code

Once having a list of the images to test the combinatorial generalization to, the idea is to change the least the original code by simply removing from the training images these test images and keeping only these images from the test ones. In such a way, differently from the original HomomorphismAE implementation we end up with two disjoint set of images between train and test. The code implementation is shown in appendix **??**.

5 Experiments and Results

In order to perform the combinatorial benchmark experiments, the following configurations have been constructed (with `[scale = 5, orientation = 14, shape = 1]` fixed):

- Recombination-to-Element

  ```
  [posX = 15, posY >= 15]
  ```

- Recombination-to-Range

  ```
  [posX >= 15, posY >= 15]
  ```

- Extrapolation

  ```
  [posX = [all], posY >= 15]
  ```

Due to the limitations expressed in the code description in the section 3 the original benchmark has been adapted to obtain increasing difficulty levels, starting from the easiest where the model cannot see a small range of images, to the hardest in which the model is not aware about all the the samples with a posY greater or equal than 15.

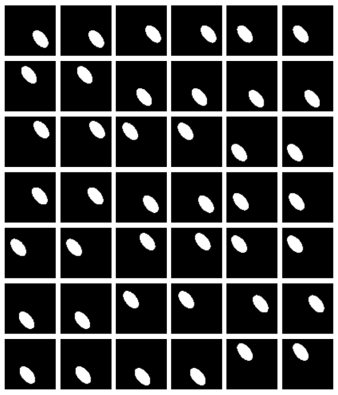

Figure 9: Reconstruction of ellipses in original setting.

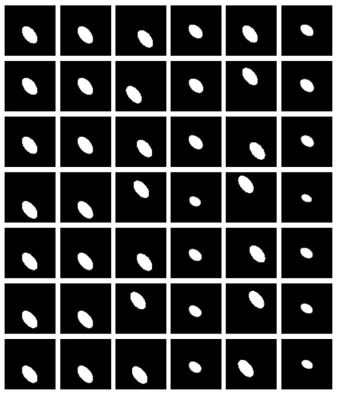

Figure 10: Reconstruction of ellipses in Recombination-to-Element setting.

In the table 5 it is possible to see the values of both train and test reconstruction loss (BCE) on the 3 benchmarks compared to the results of the model in its original setting. The performance worsening is somewhat predictable and it's proportional to the increasing difficulty of the combination method.

Another feedback is given by the actual reconstructed images, whose quality worsen consistently with the results shown in the table 5, from a perfect reconstruction in the original setting (fig. 9), to a complete overfitting and a poor reconstruction in the Extrapolation one (fig. 12).

Configuration	**Train Reconstruction Loss**	**Test Reconstruction Loss**
No-Recomb	1.68986	1.81507
Recomb-2-Element	11.06963	10.48341
Recomb-2-Range	40.11845	1300.78259
Extrapolation	468.73541	10290.81641

Table 1: Benchmark results table comparison.

Taking a look at the reconstruction plot of latent factor, we can see how structural relations between mapped input are somewhat maintained, although the clear degradation. Starting with the non recombination structure (fig. 13), where the latent projection forming a toroid, to the extrapolation one (fig. 16, where the structure seems to collapse due to the lack of datapoints.

6 Conclusion

In this project I study the problem of Combinatorial Generalization as introduced by Montero et al. (7) (6). Starting from the solution proposed by Keurti et al. (2), i.e. Homomorphism AutoEncoder, I conduct an evaluation on the benchmark proposed in (7), where 3 tests with increasing complexity are requested: *recombination to element*, *recombination to range* and *extrapolation*. Despite the good results presented in (2), the evaluation of the same model on the combinatorial generalization banchmark proposed by Montero et al. shows how the model still lacks of combinatorial capabilities, similarly to previous works. The deacrease in performance is indeed proportional to the increase in difficulty of the combinatorial test.

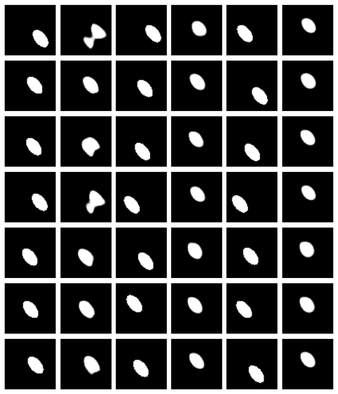

Figure 11: Reconstruction of ellipses in Recombination-to-Range setting.

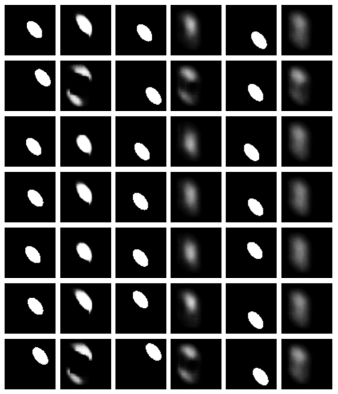

Figure 12: Reconstruction of ellipses in Extrapolation setting.

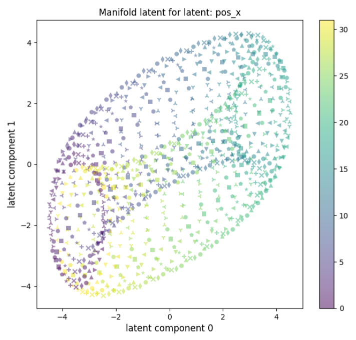

Figure 13: Latent projection without recombination.

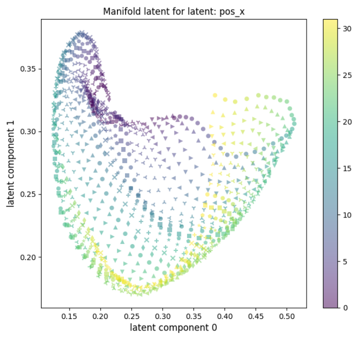

Figure 14: Latent projection in Recombination-to-Element setting.

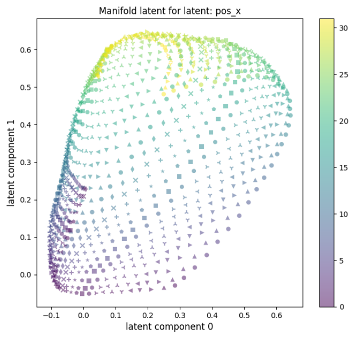

Figure 15: Latent projection in Recombination-to-Range setting.

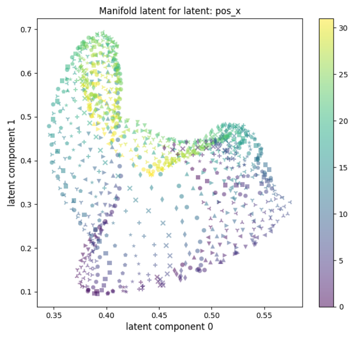

Figure 16: Latent projection in Extrapolation setting.

Acknowledgments

This paper is intended as the report of the final project of the M.Sc. course *Advance Topics in Machine Learning and Optimization*, held by Associate Professor Andrea Passerini and Post-Doc Stefano Teso at University of Trento.

References

[1] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017.

[2] Hamza Keurti, Hsiao-Ru Pan, Michel Besserve, Benjamin F Grewe, and Bernhard Schölkopf. Homomorphism autoencoder — learning group structured representations from interactions. In *UAI 2022 Workshop on Causal Representation Learning*, 2022.

[3] Hyunjik Kim and Andriy Mnih. Disentangling by factorising. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2649–2658. PMLR, 10–15 Jul 2018.

[4] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[5] Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. dsprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017.

[6] Milton L. Montero, Jeffrey Bowers, Rui Ponte Costa, Casimir JH Ludwig, and Gaurav Malhotra. Lost in latent space: Examining failures of disentangled models at combinatorial generalisation. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[7] Milton Llera Montero, Casimir JH Ludwig, Rui Ponte Costa, Gaurav Malhotra, and Jeffrey Bowers. The role of disentanglement in generalisation. In *International Conference on Learning Representations*, 2021.

A Extracting the benchmark images

The following method allows to sample from dSprite Dataset all the images that results as combinations of the given generative factors.

In the first part of the code all the input are parsed and the combination mode is identified. In the second part, for each latent, the given constraint is processed and the corresponding values/indices of the factors are saved. Then with the mapping method $\_latent\_to\_index()$, a list of indices of the images that respect those constraints is returned. Finally, the method returns the images, the indices of the images and the combination mode for further uses.

```
def exclude_from_training(imgs,
                          metadata,
                          shape=None,
                          scale=None,
                          orientation=None,
                          posX=None,
                          posY=None):
    """Extract the images given a specific set of latents value and operator.
    Example:

    >> exclude_from_training(imgs, metadata, shape="== 0", orientation="> 10")
    # Return all images of shape 0 and orientation > 10

    Latents order is ('color', 'shape', 'scale', 'orientation', 'posX', 'posY')
    Shape order is (square, ellipse, heart)
    """
    def _latent_to_index(latents, metadata):
        latents_sizes = metadata['latents_sizes']
        latents_bases = np.concatenate((latents_sizes[::-1].cumprod()[::-1][1:],
                                        np.array([1,])))
        return np.dot(latents, latents_bases).astype(int)

    num_given_factors = bool(shape) \
                      + bool(scale) \
                      + bool(orientation) \
                      + bool(posX) \
                      + bool(posY)

    factors = [None, shape, scale, orientation, posX, posY]
    factors = [">= 0" if not f else f for f in factors]
    factors = [{"compare": f.split(" ")[0], "value": int(f.split(" ")[1])}
               for f in factors]

    if num_given_factors == 5:
        combination_type = "rec2element"
    elif num_given_factors < 5 and num_given_factors >= 2:
        combination_type = "rec2range"
    elif num_given_factors == 1:
        combination_type = "extrap"
    else:
        raise ValueError("Invalid combination type")

    latents_2_sample = []

    # Color is fixed so first element is excluded
    for idx, factor_name in enumerate(metadata["latents_names"]):

        if factors[idx]["compare"] == "<":
            latents_2_sample.append(list(range(0, factors[idx]["value"])))

        elif factors[idx]["compare"] == "<=":
            latents_2_sample.append(list(range(0, factors[idx]["value"]+1)))

        elif factors[idx]["compare"] == "==":
            latents_2_sample.append([factors[idx]["value"]])

        elif factors[idx]["compare"] == ">=":
            latents_2_sample.append(list(range(factors[idx]["value"],
                                               metadata["latents_sizes"][idx])))

        elif factors[idx]["compare"] == ">":
            latents_2_sample.append(list(range(factors[idx]["value"]+1,
                                               metadata["latents_sizes"][idx])))

        else:
            raise ValueError("Invalid comparison symbol")

    latents_sampled = np.asarray(list(itertools.product(*latents_2_sample)))

    # Select images
    indices_sampled = _latent_to_index(latents_sampled, metadata)
    indices_sampled = list(set(indices_sampled))
    imgs_sampled = imgs[indices_sampled]

    return imgs_sampled, indices_sampled, combination_type
```

B Modifying homomorphismae

In the following portion of code is shown the method employed to prepare the dataset. The method is called as last operation after having built the dataset. The benchamark images are removed from the train set and are the only images kept in the test set, thus leading to two disjoint set, the former for training and the latter for testing the model. Moreover, even the transformation matrices related to the images used are removed/kept. After having cleaned the dataset, all the other steps are left unchanged to those performed by the original code.

```
def _process_dataset_with_combinatorial_indices(self):
    """Remove from train set that the triplets that contain
    at least one combinatorial image idx and keep only triplets
    of validation set that contain only combinatorial indices.

    The aim of this function is to obtain a dataset in which
    no samples from test has been seen at training time
    """

    # For train set remove the incides
    new_train_idx = []
    new_train_dj = []

    for k in range(copy.deepcopy(self.train_idx.shape[0])):
        if not set(self.all_indices[self.train_idx[k]].tolist()).\
                intersection(set(self.combinatorial_indices)):
            new_train_idx.append(self.train_idx[k].tolist())
            new_train_dj.append(self.train_dj[k].tolist())

    self.train_idx = np.asarray(new_train_idx)
    self.train_dj = np.asarray(new_train_dj)
    self.num_train = len(self.train_idx)

    # For valid set keep the indices
    new_val_idx = []
    new_val_dj = []

    for k in range(copy.deepcopy(self.val_idx.shape[0])):
        if set([self.all_indices[self.val_idx[k, 0]]]).\
                intersection(set(self.combinatorial_indices)):
            new_val_idx.append(self.val_idx[k].tolist())
            new_val_dj.append(self.val_dj[k].tolist())

    self.val_idx = np.asarray(new_val_idx)
    self.val_dj = np.asarray(new_val_dj)
    self.num_val = len(self.val_idx)
```