Learning Disentangled Representations via Mutual Information Estimation

Christian Vaona [VR495465]

August 2024

Abstract

In this project, i modified the original code to work with the SmallNORB dataset. I thought two methods to resolve the task: the classical approach used in the paper and a new method i implemented.

1 Introduction

The first method involves taking an input one image and producing two different output images: one with a colored background and the other with a colored foreground. This approach is similar to the method used for the MNIST dataset, where the digit or the background is changed.

The second method utilizes the attribute information from the SmallNORB dataset instead of the color difference. In this method, one attribute is shared, and two attributes are exclusive (one for the left and one for the right image). This report primarily focuses on the second method.

2 Methodology

2.1 Dataset Structure

The SmallNORB dataset is designed for 3D object recognition from shape. It includes images of 50 toys belonging to five generic categories: four-legged animals, human figures, airplanes, trucks, and cars. The objects were imaged by two cameras under six lighting conditions, nine elevations (ranging from 30 to 70 degrees in 5-degree increments), and eighteen azimuths (ranging from 0 to 340 in 20-degree increments).

- **Struttura delle caratteristiche :**

```
FeaturesDict({
    'image': Image(shape=(96, 96, 1), dtype=uint8),
    'image2': Image(shape=(96, 96, 1), dtype=uint8),
    'instance': ClassLabel(shape=(), dtype=int64, num_classes=10),
    'label_azimuth': ClassLabel(shape=(), dtype=int64, num_classes=18),
    'label_category': ClassLabel(shape=(), dtype=int64, num_classes=5),
    'label_elevation': ClassLabel(shape=(), dtype=int64, num_classes=9),
    'label_lighting': ClassLabel(shape=(), dtype=int64, num_classes=6),
})
```

Figure 1: Image that show the SmallNORB dataset structure

Unlike MNIST, which consists of single images with only the attribute for the digit category, SmallNORB provides pairs of images that differ slightly [Figure 1]. Initially, i considered processing these image pairs as inputs. However, i found that the additional attribute information in the `-info` file applied only to one image of each pair, leaving the other image without relevant attribute information [Figure 2].

	image	image2	instance	label_azimuth	label_category	label_elevation	label_lighting
8			9	10	2 (airplanes)	2	1

Figure 2: Example image for the dataset image pair attributes with information to only the first of them

After those consideration i have decided to make the category shared and to make two of the other attribute exclusive. To address this, i developed a method to create image pairs that share the same category but differ by one attribute each, extracted from the original dataset. I focused on the first image of each existing pair, where attribute information were available. The code generates a lookup dictionary mapping combinations of categories and attributes to image indices. This allows the creation of unique image pairs by varying two attributes and ensures no duplicate pairs exist. This data are stored as torch tensors in a dictionary format to be processed later as input.

The exclusive attributes chosen for experimentation are elevation and lighting, which i will discuss further in the testing and results section.

2.2 Code Modifications

2.2.1 SDIM – Shared Dimension

For the SDIM component, i added classifiers for the category, elevation, and lighting attributes. This involved implementing the gradient calculation, loss, accuracy, and optimizer. Additionally, i modified the `img_feature_size` in `SDIM.py` to match the feature size of the new images, as it was hard-coded rather than calculated dynamically.

2.2.2 EDIM – Exclusive Dimension

The EDIM component presented several challenges. The first was selecting among the available attributes (lighting, elevation, and azimuth). Initially, i chose azimuth, but its three-dimensional complexity resulted in low accuracy, leading me to suspect an implementation issue. After some test i eventually switched to elevation, which provided sufficient results.

Another challenge was computing accuracy for the exclusive attributes in the image pairs. Initially, I considered adding the accuracy for the same attribute in the left and right images, similar to the category. However, since the category is shared and not learned by the EDIM model, i decided to keep it separate for the left and right images. I then coded the elevation and lighting to be the exclusive attributes, requiring only one of them per image. Specifically, the elevation attribute was assigned to the left image and the lighting attribute to the right image.

2.2.3 EDIM Config file – Exclusive Dimension

For the exclusive attributes, the results were poor in terms of accuracy, so i increased the number of epochs from 11 to 16 and the learning rate from 0.0001 to 0.0003 to enhance the performance.

3 Testing and Results

During various tests with changing exclusive attributes, i noticed that the lighting attribute was the easiest to learn, as it is a low-level feature. In contrast, elevation and azimuth where difficult to learn due to their three-dimensional properties. To improve accuracy for these attributes, enhancing the encoder would be crucial. A better encoder would ensure that the output features are relevant, high-quality, and discriminative, significantly aiding the classifier. However, to remain consistent with the paper, i decided not to modify the encoder and conducted tests using the provided code adjusting the parameter in the exclusive configuration.

In the end, i discarded the azimuth attribute and chose elevation, achieving acceptable accuracy. The shared category attribute reached 93% accuracy, while the exclusive attributes of elevation and lighting achieved 74% and 98%,

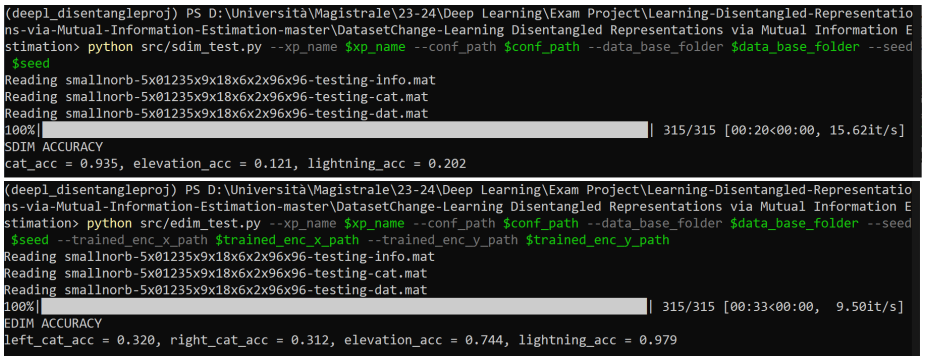

Figure 3: Model result

respectively. The elevation result is lower than the paper's benchmark, but improvements could be made by enhancing the encoder like i explained before.

4 Running the Project

After setting up the environment and installing the necessary requirements, use the train command in the README-SmallNORB to run the project to train and the test command to test the model and display the result.

After the train i save also the model SMDI.pth and EDIM.pth in the main folder to use them later for the test, so if you start again a new train this models will be overwritten.

Note: Remember to change the path to the pretrained encoder of domains X and Y. I have also added a seed = 42 to make the run reproducible.