Lab 6 report

Lucie Sénéclauze Bertolo

May 6, 2025

1 Introduction

In this report, I describe the implementation and evaluation of a Conditional Denoising Diffusion Probabilistic Model (DDPM) for generating synthetic images from multi-label textual descriptions. As a starting point, I used the publicly available model `byrkbrk/conditional-ddpm`, initially designed for single-label conditioning on the MNIST dataset. I adapted this model to handle multi-label inputs and applied it to the i-CLEVR dataset, a synthetic dataset of 3D object scenes. In the first part of the report, I detail the modifications made to the original codebase, including the change of dataset, the addition of a multi-label conditioning mechanism, and the implementation of various noise schedules. I also present the training pipeline and preprocessing steps. In the second part of the report, I evaluate the model using a provided classifier. Finally, I analyze the results, showcase generated images, and discuss data augmentation techniques.

2 Implementation details

As a starting point for my model, I used the publicly available model byrkbrk/conditional-ddpm (available here). I then adapted it to support multi-label conditioning (it was made for single-label conditioning), changed the dataset from MNIST to i-clevr and added a testing function using the evaluator we were provided with.

2.1 Data preprocessing

I created a *test_dataloader* as well as a *train_dataloader* to load the images to the model. They resize the images to dimension (64,64) and then normalise them with *transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))*as to be compatible with the input of the given pretrained evaluator.

2.2 Training pipeline

The training pipeline is a usual training pipeline for a DDPM as seen in figure 1. We sample noise accordingly to the noise schedule and add it to the image. Then we take the noisy image and its context (the objects that were on the original image and their colors) and feed it to a UNET to try to predict the shape of the noise, thus teaching the UNET to remove noise from an image while knowing what was on the non-noisy image. Finally, we compute the MSE loss between the noise and the estimated noise outputted by the UNET and backpropagate it to update the network's weights.

2.3 DDPM noise schedule

In DDPM, the noise schedule determines how much noisy the image will be at each instant. Then during training, we sample randomly a time t and add noise accordingly to the noise schedule at time t. The most common noise schedule, and also the one that was originaly implemented in the yrkbrk/conditional-ddpm code, is a linearly increasing one. I also implemented a cosine noise (inspired by this paper) as well as a quadratic one. Figure 2 shows the running average of the losses for three runs using the three different noise schedules. As we can see, all of them seen eligible candidates as the loss is decreasing over the epochs. To choose the best one, I evaluated them using the given evaluator over the *test.json*. Figure 3 shows the comparison of the accuracy over the new_test dataset for the 100 epochs checkpoint. For my final model, I chose the linear DDPM noise schedule as it gave me the best accuracy.

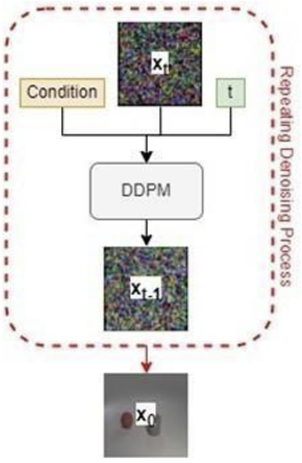


Figure 1: Training pipeline

Source: Lab6 description

Figure 2: Loss comparison

```
Accuracy comparison for linear, cosine and quadratic DDPM noise schedule
linear : 0.6510
cosine : 0.5104
quadratic : 0.4427
```

Figure 3: Accuracy comparison

3 Results and discussion

3.1 Synthetic image grid

Figure 4 shows the synthetic images generated using the descriptions in the *test.json* file. Figure 5 shows the best classification accuracy that I had for those images.
Figure 6 shows the synthetic images generated using the descriptions in the *new_test.json*. Figure 7 shows the best classification accuracy that I had for those images.

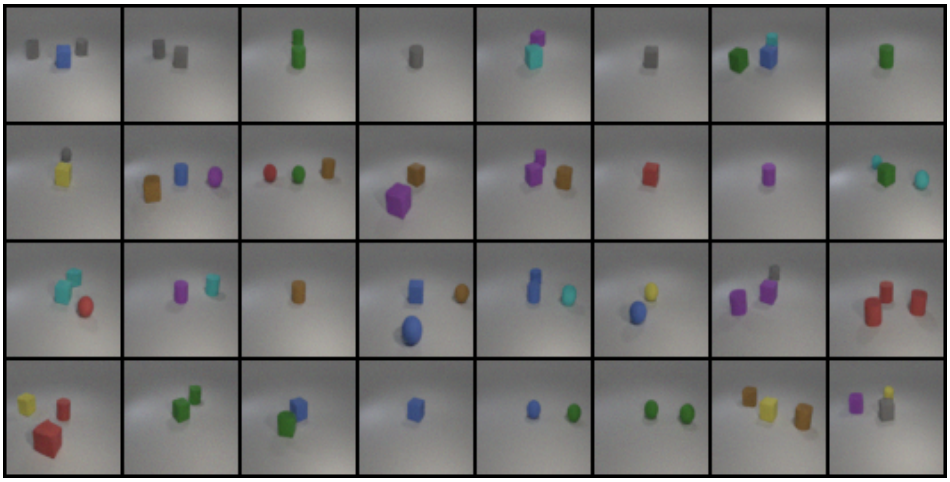


Figure 4: Synthetic image grid with *test* testing data

```
Classification accuracy on the data in the test.json file :
0.671875
```

Figure 5: Accuracy for the *test* testing data

Figure 6: Synthetic image grid with *new_test* testing data

```
Classification accuracy on the data in the new_test.json file :
0.61458333
```

Figure 7: Accuracy for the *new_test* testing data

3.2 Denoising process

In figure 8 , we can see the denoising process for an image with the label [”red sphere”, ”cyan cylinder”, ”cyan cube”].

3.3 Data augmentation

Since the dataset is rather small (18009 images), I decided to perform data-augmentation. I considered different transformations but got rid of several of them because they would have changed the type of the data :

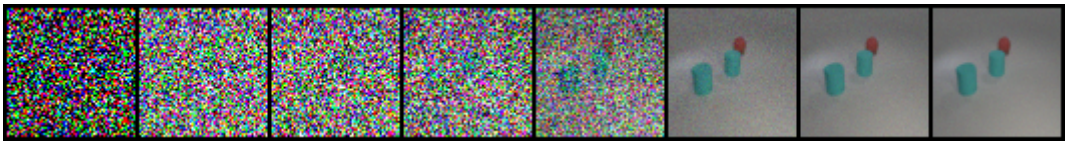


Figure 8: Denoising process

- **Zooming in on the image**, see figure 9. However, in the training set all the objects have roughly the same size so zooming in on them would have created a more difficult dataset.
- **Rotating the image at different angles**, see figure 10. However, on each image there is a "lightning" phenomenon with the bottom of the image being more clear than the top (like if it was lit with a lamp above) so if I had performed rotations, I would have moved the lightning and made the dataset more difficult.
- **Changing the colors**, see figure 11. I have thought about changing the colors by taking the negative value of the image. That way, I would know what each color would become (eg. "red cube" would become "cyan cube"). It would only work for the red, cyan, yellow and blue shapes but I could restrict the recoloring to only these colors. Another problem would have been the fact that the background would have changed color but to fix this, I could have trained a model to do binary segmentation and only applied the color reversal to the shape and not the background. I did not implement this for lack of time.

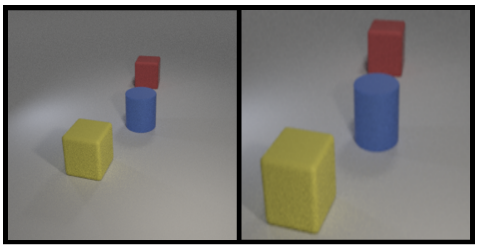


Figure 9: Zooming in on the image

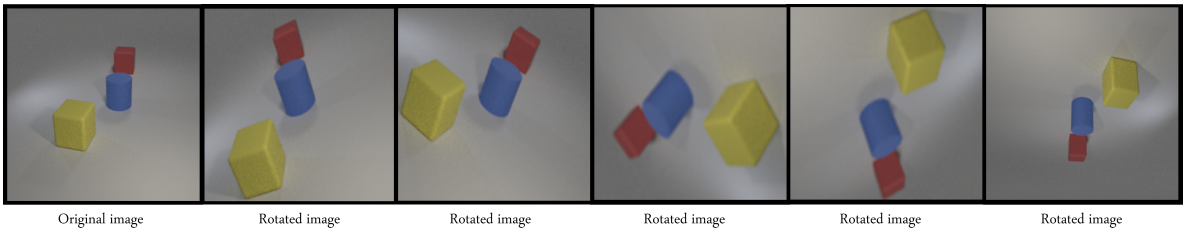


Figure 10: Rotating the image

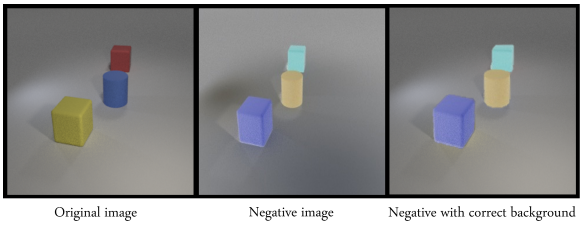
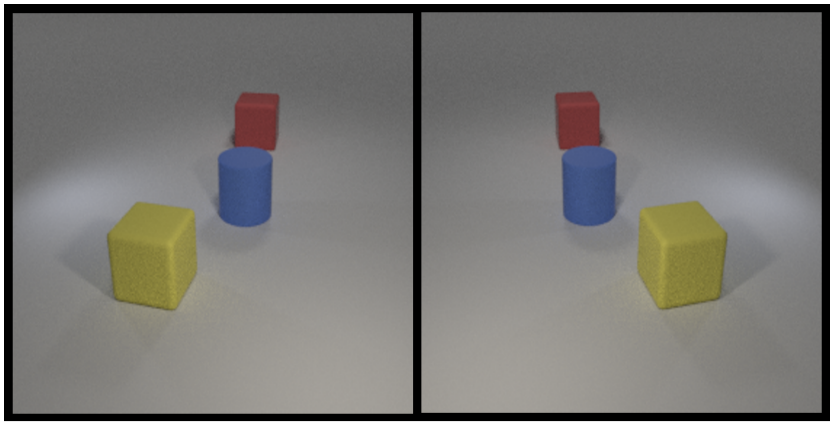


Figure 11: Changing the colors

For data-augmentation I decided to mirror all the images as I thought it was the best augmentation that also allowed me to preserve the original shape of the dataset as well as its simplicity (see figure 12).



Original image Mirrored image

Figure 12: Mirroring the image

To evaluate the utility of the data augmentation, I used the provided evaluator. Figure shows the accuracy on the *test.json* of both non-augmented and augmented models. Since the augmented model receives twice as much data, the evaluation is made at epoch 44 for the augmented but 88 for the non augmented. Figure 3 shows the comparison between the two accuracies. As we can see, data augmentation didn't improve the model's performance.

```
Accuracy comparison for non augmented and augmented datasets
augmented dataset (epoch 44) :  0.6771
non augmented dataset (epoch 88) : 0.6708
```

Figure 13: Accuracy comparison

4 Conclusion

In this report, I explained my implementation of a Conditional Denoising Diffusion Probabilistic Model. I tried to determine which type of noise schedule was the best and saw that a linear noise schedule yielded better results. I have also implemented data augmentation but found that it did not improve the model's performances.