Implementing unpaired learning with Optimal Transport on 3D MNIST images

Nikita Sushko [1] **Mikhail Shvetsov** [1] **Artem Chemodanov** [1] **Roman Khalikov** [1] **Denis Kukushkin** [1]

Abstract

Computational Optimal Transport (OT) is a set of tools, which provide a way of transforming one distribution into another with the minimal effort. Recently scalable neural OT based methods, which do not require paired training dataset, have been developed and applied to a wide range of tasks, including transfer learning, generative modeling and super-resolution.

In this project, we are to implement and train OT approach on 3D MNIST digits dataset, analyze the applicability of OT method to 3D image-to-image translation.

Github repo: https://github.com/chameleon-lizard/ML_Project

1. Introduction

Solving optimal transport problems with neural networks has become widespread in machine learning tentatively starting with the introduction of the large-scale OT(Seguy et al., 2018) and Wasserstein Generative Adversarial Networks(Arjovsky et al., 2017). They all have one thing in common – they use the transport cost as loss. However, recently it was demonstrated that OT plan can be used as the generative model in itself(Rout et al., 2022).

In the 2023 paper Neural Optimal Transport(Korotin et al., 2023) the possibility of using neural networks to learn the optimal transport map for 2D image generation was explored.

In this project, we were to explore the possibility of using neural networks to learn the optimal transport maps for 3D images by using the 3D MNIST dataset as an example.

What we were to do in the project:

- Study the related literature on optimal transport;
- Fetch 3D MNIST dataset, split train images on trainA (digits '3'), trainB (digits '5'). Perform testing on corresponding digits ('3') from test part;
- Change the code of Neural Optimal Transport, e.g., use architectures of the transport map $T_\theta : 3 \times R^{3\times16\times16\times16} \to R^{3\times16\times16\times16}$, potential $f_w : 3 \times R^{3\times16\times16\times16} \to R$, to make it applicable to colored 3D images;
- Learn an OT mapping between trainA and trainB samples using strong quadratic cost function. Save the final checkpoints (weights θ of the map T);
- Select 5 random digits from trainA, show the corresponding OT mapped digits. Visualize the results.
- Analyze the results. Do the output images resemble the shape of the input images shape? Does the map preserve the color of the digit during the translation? Does the quadratic cost turn to be a good choice for this pair of datasets? Yes/No? Why?

The main contributions of this report are as follows:

- Created a new flavour of 3D MNIST dataset;
- Implemented 3D ResNet and 3D UNet architectures as the transport and potential networks;
- Studied applicability of direct and transfer learning of the transport map on two different datasets;

2. Related work

Previously to learn generative models an OT cost loss function was used. For example Wasserstein GAN (Arjovsky et al., 2017) are the popular generative models built on the theory of OT and the Kantorovich duality. This is the models with good stability of learning, without problems like mode collapse, they provides meaningful learning. However it was stated (Korotin et al., 2022) that WGAN Dual OT solvers should not be considered as a meaningful estimators of Wasserstein-1 distance $\mathbb{W}_1$ as they exhibit large error but still can be used for minimization in variational problems.

[1]Skolkovo Institute of Science and Technology, Moscow, Russia. Correspondence to: Nikita Sushko <Nikita.Sushko@skoltech.ru>.

Final Projects of the Machine Learning 2020 Course, Skoltech, Moscow, Russian Federation, 2020.

Another approach (Seguy et al., 2018) allows to compute OT plan for one-to-many map between two distributions. It introduced the first tractable algorithms for computing both the regularized OT objective and optimal maps in large-scale or continuous settings The problem is that the entropy regularized OT recover regularized OT plan that is biased from the true one, it is hard to sample from it or compute its density. (Korotin et al., 2023)

A new approach (Korotin et al., 2023) in OT problem solving was proposed recently. The main idea is to compute deterministic and stochastic OT plans by solving the dual problem of the weak optimal OT cost formulation as a saddle point optimization problem. It generalizes previously known scalable approaches and provides a better interpretability of the learned map and allows to control the amount of diversity in generated samples. This method extracts the stochastic OT map from the solution however it may not be the optimal stochastic map which may lead to the issues. In this paper this approach will be tested on 3D MNIST images.

3. Preliminaries

To better understand the following work, the strong OT formulation should be established.

3.1. Strong OT formulation

For $P \in P(X), Q \in P(Y)$ and a cost function $c : X \times Y \to R$, Monge's primal formulation of the OT cost is

$$Cost(P, Q) = inf_{T_{\#}P=Q} \int_X c(x, T(x))dP(x),$$

where the minimum is taken over measurable functions $T : X \to Y$ that map P to Q. The optimal T^* is called the OT map.

Since this formulation is not symmetric and does not allow mass splitting, i.e. for some $P, Q \in P(X), P(Y)$ there may be no T satisfying $T_{\#}P = Q$. Thus, Kantorovitch proposed the following relaxation:

$$Cost(P, Q) = inf_{\pi \in \Pi(P,Q)} \int_{X \times Y} c(x, y)d\pi(x),$$

where minimum is taken over all transport plans π, , i.e., distributions on $X \times Y$ whose marginals are P and Q. The optimal $\pi^* \in \Pi(P, Q)$ is called the optimal transport plan. If π^* is of the form $[id, T]_{\#P \in \Pi(P,Q)}$ for some T^*, then T^* minimizes the formula in the Monge's formulation. In this case, the plan is called deterministic. Otherwise, it is called stochastic (nondeterministic).

4. Algorithms and Models

To find transport map T from distribution P to distribution Q we need two networks: the Transport network (a generator like network that computes the transport map) and the Potential network (a discriminator-like network). For our project, we've used two networks for 3D data of the shape $3 \times 16 \times 16 \times 16$: modified UNet and ResNet.

For the transport network T, we've decided to use a shallower version of UNet, based on implementation from Alex Korotin's GitHub repository with Neural Optimal Transport. This helped our model to learn a bit quicker, since learning transport maps is quite a resource intensive task. We've also adapted it to work with 3D data by changing the 3D convolutions with 2D convolutions.

For the potential network, we've taken a ResNet like network with 3D convolutions instead of 2D convolutions.

For experiments with 2D images, we've decided to stick with the same network that Alex Korotin used in his Neural Optimal Transport seminar dedicated to the strong OT with quadratic cost.

5. Datasets

We used two datasets to train our models:

- 3D MNIST dataset
- 2D MNIST converted into 3D

5.1. 3D MNIST from Kaggle

This dataset is represented by point clouds created by the original MNIST images. The dataset is stored as 4096 dimentional vectors that were obtained from the voxelization (x:16, y:16, z:16) of all the 3D point clouds. There are 10000 train images in dataset, but due to the fact that we were working with specific numbers we were limited to only $\frac{1}{10}$ of the dataset. Along with the original digits in the dataset there are rotated and noisy images. To color the dataset we processed it with a simple function that expanded dimensions and assigned RGB colors to each point. We provide a link to the colored dataset along with the corresponding code on our GitHub.

5.2. 2D MNIST converted into 3D

We considered that due to a lot of noise and rotation it was quite hard to construct a right mapping for our model when being trained on the Kaggle dataset. Therefore we decided to create our own dataset from the 2D MNIST images. To do so we stacked several layers of 2D images thus getting (x:16, y:16, z:16) voxelization. With this dataset we also had an opportunity to vary the amount of transforms applied

to the dataset. We used random rotations and a coloring function originally devised for 2D MNIST. The processing functions can be found on our GitHub.

6. Experiments and Results

As the project proposal suggests, we needed to explore Neural Optimal Transport on Kaggle 3D MNIST dataset by adding random colors to images and then trying to find a transport map that converts threes to fives without changing their color. We've done six experiments on two different datasets to explore the possibility of doing that.

6.1. Transporting images from colored Kaggle's 3D MNIST dataset

The first experiment would be to test, if a basic voxel-to-voxel model, that we can find the transport map that transports randomly colored threes from Kaggle's 3D MNIST dataset to fives of the same color. Unfortunately, the dataset size was pretty small to begin with and the amount of threes and fives was less than a thousand. In addition to a small size of the dataset, the numbers inside were given rotations with three degrees of freedom. Normally, this wouldn't be a problem, since polygon-based models are rotationally invariant, but the since the images inside the dataset were comprised of voxels, which are representing basically a grid of 3D pixels, rotations would break the lines in the original data and sometimes make the images unusably distorted.

We've run the training loop for the UNet and ResNet networks for 10001 iterations with 10 iterations of the transport network training between every iteration of the potential network training. We've used the Adam optimizer for both networks with Learning Rate set to 10^{-5} for UNet and 10^{-5} for ResNet.

The training was carried out on Intel Core i7 7700K, 16 GB of RAM and NVidia RTX 3070ti. It took about 2 seconds to do one iteration, the whole training process took about 2 hours.

The model could not converge in given amount of iterations and even failed to learn the colors of the transported numbers. We've had two theories about what lead to failure in convergence that we wanted to test:

1. Failure of convergence was the result of a small dataset size and the amount of same colored subsamples was not enough for the model to learn;
2. Rotations of the images introduced too much complexity to the images, which seriously hindered model's ability to find the transport map;

To test these two theories another experiments were carried out.

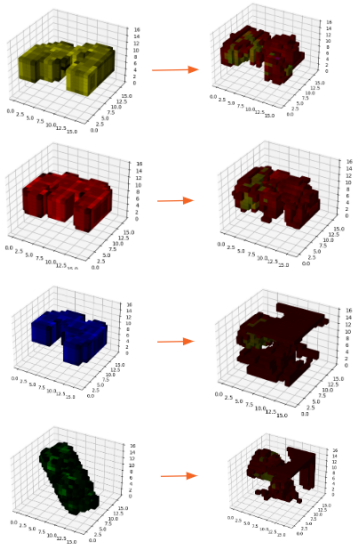

Figure 1. Results of OT on the colored Kaggle's 3D MNIST dataset

6.2. Transporting images from uncolored Kaggle's 3D MNIST dataset

In order to test the first theory, we've first tried to find the transport map on a bigger dataset. By using uncolored data, we would have bigger sets of the same color numbers, so the model theoretically would have easier time determining what features to transport and what features to leave.

Just as before, we've run the training loop for the UNet and ResNet networks for 10001 iterations with 10 iterations of the transport network training between every iteration of the potential network training. We've used the Adam optimizer for both networks with Learning Rate set to 10^{-5} for UNet and 10^{-5} for ResNet.

The training was carried out on AMD Ryzen 3900, 32 GB of RAM and NVidia RTX 3070ti. It took about 3 seconds to do one iteration, the whole training process took about two hours.

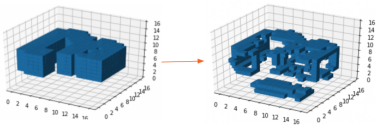

Figure 2. Results of OT on the uncolored Kaggle's 3D MNIST dataset

The model could not converge in given amount of iterations. The spatial information was lost, so the images were completely unusable. Judging from loss values, the model started fine, but then quickly diverged.

6.3. Transporting images from our unrotated colored 3D MNIST dataset

To further test the first theory, creation of our own dataset was proposed. Since MNIST contains 50000 labeled training images, we could embed colorization, random rotation and 3D-ification the images into the TorchVision's transform pipeline. In this experiment, we've only applied colorization and 3D-fication of the images from MNIST, without rotation. This lead to vastly increased dataset size – from around 650 images of threes to around 6500 images.

Just as before, we've run the training loop for the UNet and ResNet networks for 30001 iterations with 10 iterations of the transport network training between every iteration of the potential network training. We've used the Adam optimizer for both networks with Learning Rate set to 10^{-5} for UNet and 10^{-5} for ResNet.

The training was carried out on Intel Core i9 12900KF, 64 GB of RAM and NVidia RTX A5000. It took about 1.2 seconds to do one iteration, the whole training process took about six hours.

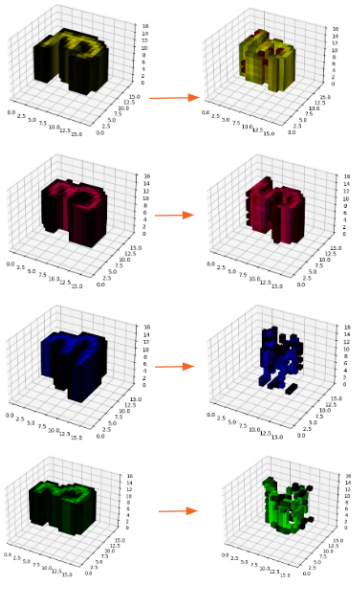

Figure 3. Results of OT on our 3D conversion of MNIST dataset, without rotation

The model successfully learned the mapping between the yellow and red colored digits, while completely losing spatial information in case of blue and green digits. The colors, however, were preserved.

6.4. Transporting images from rotated colored 2D MNIST dataset

The semi-success of the third experiment meant that the transport map could be learned with a bigger dataset and without rotations. The next logical step would be to test if it could be learned with rotated data.

To test this theory, we've taken the Jupyter Notebook of Alex Korotin's seminar on Neural Optimal Transport on the 2D MNIST images and added random rotations to the data preprocessing pipeline.

The training loop was run 8001 times, with 5 iterations of the transport network training between every iteration of the potential network training. We've used the Adam optimizer for both networks with Learning Rate set to 10^{-4} for UNet and 10^{-5} for ResNet.

The training was carried out Google Colab free tier hardware. It took about two hours for the whole training loop to be ran.

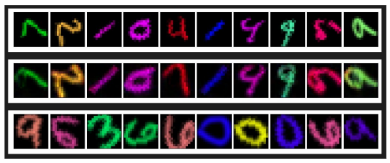

Figure 4. Results of OT on rotated 2D MNIST

The model could not learn the mapping between two different digit styles. In the best case, the images stayed exactly the same, in the worst case, they started to disappear.

6.5. Transporting images from our rotated colored 3D MNIST dataset

The results of the previous experiments suggested that learning the transport map from the rotated dataset would be much harder than learning with the unrotated dataset on 2D data. To further increase our confidence in this hypothesis, we decided to rerun the same experiment on the rotated with one degree of freedom 3D MNIST data.

To get the rotated 3D data, we simply inserted the random rotation inside the TorchVision transforms before the colorization and 3D-ification. This meant that our data is rotated with only one degree of freedom, not with three degrees of freedom as in the Kaggle's 3D dataset.

We've run the training loop for the UNet and ResNet networks for 4001 iterations with 10 iterations of the transport network training between every iteration of the potential network training. We've used the Adam optimizer for both networks with Learning Rate set to 10^{-4} and 10^{-5} for UNet and 10^{-4} for ResNet.

The training was carried out on Intel Core i9 12900KF, 64 GB of RAM and NVidia RTX A5000. It took about 1.4 seconds to do one iteration, the whole training process took about 40 minutes.

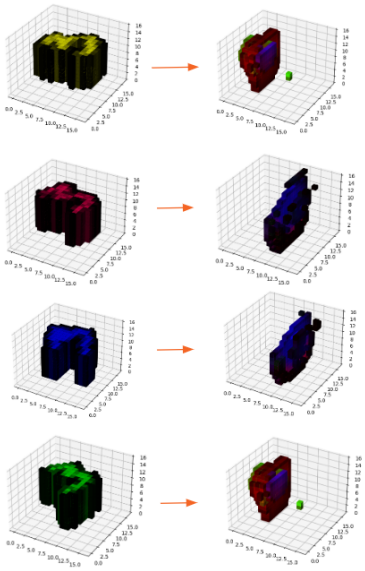

Figure 5. Results of OT on our rotated 3D MNIST

The model failed to learn the transport map. We can see signs of mode collapse – the results for yellow-green and blue-red sets of numbers are the same or really close.

6.6. Transporting images from our rotated colored 3D MNIST dataset using transfer learning

The next experiment was designed to test the applicability of transfer learning in the task of Optimal Transport for 3D images. We've ran the experiment with unrotated 3D MNIST data for 30000 iterations and tried to use learned weights as the starting points for another set of data.

We've run the training loop for the UNet and ResNet networks for 5001 iterations with 10 iterations of the transport network training between every iteration of the potential network training. We've used the Adam optimizer for both networks with Learning Rate set to 10^{-4} and 10^{-5} for UNet and 10^{-4} for ResNet.

The training was carried out on Intel Core i9 12900KF, 64 GB of RAM and NVidia RTX A5000. It took about 1.4 seconds to do one iteration, the whole training process took about 40 minutes.

Again, the model semi-successfully learned the transfer map for yellow and red numbers, but failed to learn anything for the blue and green numbers. However, in all cases, it still retained the same colors.

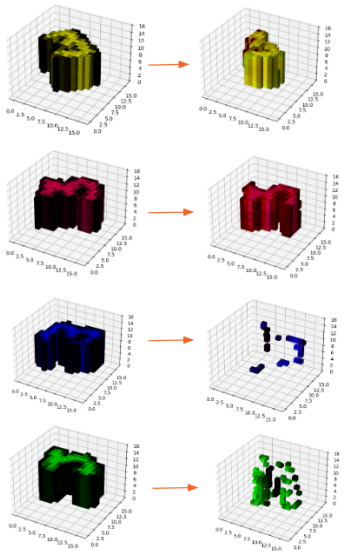

Figure 6. Results of transfer learning of OT on our rotated 3D MNIST

6.7. Transporting images from Kaggle's colored 3D MNIST dataset using transfer learning

The next experiment was designed to test the applicability of the same weights to do the task of Optimal Transport of the Kaggle's 3D data. We've taken the weights from 30000 iteration training on unrotated 3D MNIST and used these weights as the starting point.

We've run the training loop for the UNet and ResNet networks for 5001 iterations with 10 iterations of the transport network training between every iteration of the potential network training. We've used the Adam optimizer for both networks with Learning Rate set to 10^{-4} and 10^{-5} for UNet and 10^{-4} for ResNet.

The training was carried out on Intel Core i9 12900KF, 64 GB of RAM and NVidia RTX A5000. It took about 1.4 seconds to do one iteration, the whole training process took about 45 minutes.

The model diverged in the first 30 iterations, failing to learn anything. Curiously, it exhibited the same results as in first experiment with colored 3D MNIST data, without pretraining on unrotated data.

6.8. Transporting images from Kaggle's colored 3D MNIST dataset using pretraining on the rotated data

The next experiment was designed to test the applicability of the weights from the successful transfer learning experiment from our rotated 3D MNIST data to the Kaggle's 3D MNIST data. We've used the 3000 iteration checkpoint to initialize

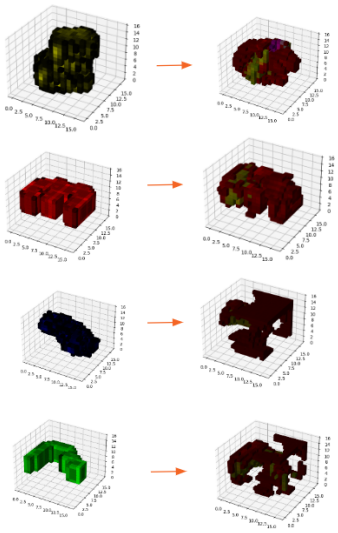

Figure 7. Results of transfer learning of from unrotated 3D MNIST to Kaggle's 3D MNIST data

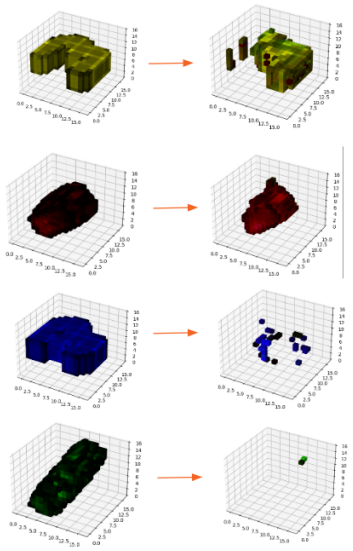

Figure 8. Results of transfer learning of from unrotated 3D MNIST to Kaggle's 3D MNIST data

the weights of our network and continue the training on the 3D Kaggle set of data.

We've run the training loop for the UNet and ResNet networks for 1001 iterations with 10 iterations of the transport network training between every iteration of the potential network training. We've used the Adam optimizer for both networks with Learning Rate set to 10^{-4} and 10^{-5} for UNet and 10^{-4} for ResNet.

The training was carried out on Intel Core i9 12900KF, 64 GB of RAM and NVidia RTX A5000. It took about 1.4 seconds to do one iteration, the whole training process took about 20 minutes.

Again, the training diverged in the first 10 iterations, but it curiously exhibited an ability to somewhat work on the 3D MNIST data, still maintaining the same color, but losing even more data in the green and blue digits and, while not quite giving out the correct fives on the red and yellow data, but still having some resemblense to digits, not a complete mess.

Algorithm 1 2D image to 3D

Input: $image2D_{3\times16\times6}$
$Image3D \leftarrow zeros(3, 16, 16, 16)$
for $i = 1$ **to** 16 **do**
  if $i \leq 4$ or $i \geq 12$ **then**
    $Image3D[..., i] \leftarrow$ zeros like $(image2D)$
  else
    $Image3D[..., i] \leftarrow image2D$
  end if
end for
return $Image3D$

7. Conclusions

The experiments proved that transfer learning on the 3D data is indeed possible, but a lot harder, than on 2D data. The most fascinating discovery was applicability of transfer learning techniques for 3D OT with different degrees of freedom of rotation.

The strong quadratic cost showed to be an adequate cost for the OT algorithm on 3D data. However, we think that it might not be enough for because of extra intrinsic dimensionality of the Kaggle's 3D MNIST dataset that comes from more degrees of freedom in rotation.

One of the directions of the future work would be to test different models for the OT and applying weights of said models to estimate the transfer map as described in the 8-th subsection of **Experiments** section. Also, other cost functions have to be evaluated on all of the proposed data, to see which will work best.

Kaggle's 3D MNIST dataset proved to be too small for the task of Neural Optimal Transport. Thus, another result of this project is the development of another flavour of 3D MNIST dataset. One of other directions of future work would be to establish a stable pipeline to introduce more degrees of freedom to random rotations of the images.

Algorithm 2 3D MNIST colorization

Input: images $x_{16\times6\times16}, sizem$
for $i = 1tom$ **do**
  $color \leftarrow random('red','green','blue','yellow')$
  $ColoredImage \leftarrow zeros(3, 16, 16, 16)$
  if $color = red$ **then**
    $ColoredImage[0, ...] = (V[...] > 0) * 255$
    $ColoredImage[1, ...] = V$
    $ColoredImage[2, ...] = V$
  end if
  if $color = yelow$ **then**
    $ColoredImage[0, ...] = (V[...] > 0) * 255$
    $ColoredImage[1, ...] = (V[...] > 0) * 255$
    $ColoredImage[2, ...] = V$
  end if
  if $color = green$ **then**
    $ColoredImage[0, ...] = V$
    $ColoredImage[1, ...] = (V[...] > 0) * 255$
    $ColoredImage[2, ...] = V$
  end if
  if $color = blue$ **then**
    $ColoredImage[0, ...] = V$
    $ColoredImage[1, ...] = V$
    $ColoredImage[2, ...] = (V[...] > 0) * 255$
  end if
end for
return $ColoredImage$

Algorithm 3 Train Network

$T \leftarrow UNET(3, 3)$
$f \leftarrow Resnet(16, nc = 3)$
$steps \leftarrow 4000$
Quadratic Loss $\leftarrow (X - Y)^2$
$f_Loss \leftarrow (X - Y)^2$
for $i = step$ **to** steps **do**
  for $i = 1$ **to** 10 **do**
    Sample X
    Send data through the T network: $T_X \leftarrow T(X)$
    Calculate Quadratic Loss for X and T_X
    Zero the gradients
    Calculateupdate the gradients for T with backprop
    Optimize one step
  end for
  Set training mode to f network and evaluation mode to T network
  Send data through the T network
  Sample X and Y
  Calculate loss for f: $f(T_X) - f(Y)$
  Zero the gradients
  Calculateupdate the gradients for f
end for

References

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein GAN, 2017.

Korotin, A., Kolesov, A., and Burnaev, E. Kantorovich Strikes Back! Wasserstein GANs are not Optimal Transport? In *36th Conference on Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=VtEEpi-dGlt.

Korotin, A., Selikhanovych, D., and Burnaev, E. Neural optimal transport. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=d8CBRlWNkqH.

Rout, L., Korotin, A., and Burnaev, E. Generative modeling with optimal transport maps. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=5JdLZg346Lw.

Seguy, V., Damodaran, B. B., Flamary, R., Courty, N., Rolet, A., and Blondel, M. Large-scale optimal transport and mapping estimation, 2018.

A. Team member's contributions

Explicitly stated contributions of each team member to the final project.

Nikita Sushko (30% of work)

- Reviewing literate on the topic
- Transforming the UNet and ResNet to work with 3D images
- Implementing the plotting function for voxel images
- Experimenting with model parameters on Kaggle 3D MNIST and our 3D MNIST dataset

Mikhail Shvetsov (20% of work)

- Prepare and Colorize 3D MNIST Dataset
- Made 3D images from 2D MNIST Dataset

Denis Kukushkin (20% of work)

- Ran experiments
- Helped implementing colorization, 3D-fication and plotting of the images
- Written the paper
- Created the presentation

Artem Chemodanov (15% of work)

- Ran experiments in Colab
- Written the paper
- Created the presentation

Roman Khalikov (15% of work)

- Ran experiments
- Existing papers research
- Written the paper
- Created the presentation

B. Reproducibility checklist

Answer the questions of following reproducibility checklist. If necessary, you may leave a comment.

1. A ready code was used in this project, e.g. for replication project the code from the corresponding paper was used.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   General comment: If the answer is **yes**, students must explicitly clarify to which extent (e.g. which percentage of your code did you write on your own?) and which code was used.

   Students' comment: The code for UNet and ResNet was based on 2D realization of the same networks from Alex Korotin's GitHub. The code for 2D image colorization was taken from Alex Korotin's GitHub. The code for voxel image plotting was based on MatPlotLib's Axes3D examples documentation page.

2. A clear description of the mathematical setting, algorithm, and/or model is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

3. A link to a downloadable source code, with specification of all dependencies, including external libraries is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

4. A complete description of the data collection process, including sample size, is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

5. A link to a downloadable version of the dataset or simulation environment is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

6. An explanation of any data that were excluded, description of any pre-processing step are included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

7. An explanation of how samples were allocated for training, validation and testing is included in the report.

   ☐ Yes.
   ☐ No.
   ☑ Not applicable.

   Students' comment: None

8. The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results are included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: Hyperparameter search was done by hand.

9. The exact number of evaluation runs is included.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

10. A description of how experiments have been conducted is included.

    ☑ Yes.
    ☐ No.
    ☐ Not applicable.

    Students' comment: None

11. A clear definition of the specific measure or statistics used to report results is included in the report.

    ☐ Yes.
    ☐ No.
    ☑ Not applicable.

    Students' comment: Since most of the scores that are used to test the generative model qualities are based on the results of the inference of pretrained 2D neural networks, not applicable.

12. Clearly defined error bars are included in the report.

    - ☐ Yes.
    - ☐ No.
    - ☑ Not applicable.

    Students' comment: None

13. A description of the computing infrastructure used is included in the report.

    - ☑ Yes.
    - ☐ No.
    - ☐ Not applicable.

    Students' comment: We've used three different servers for model training. You can find the exact parameters in the "experiments" section.