DDPM Image To Image Translation

Alexander Sharshavin and Elfat Sabitov

Introduction

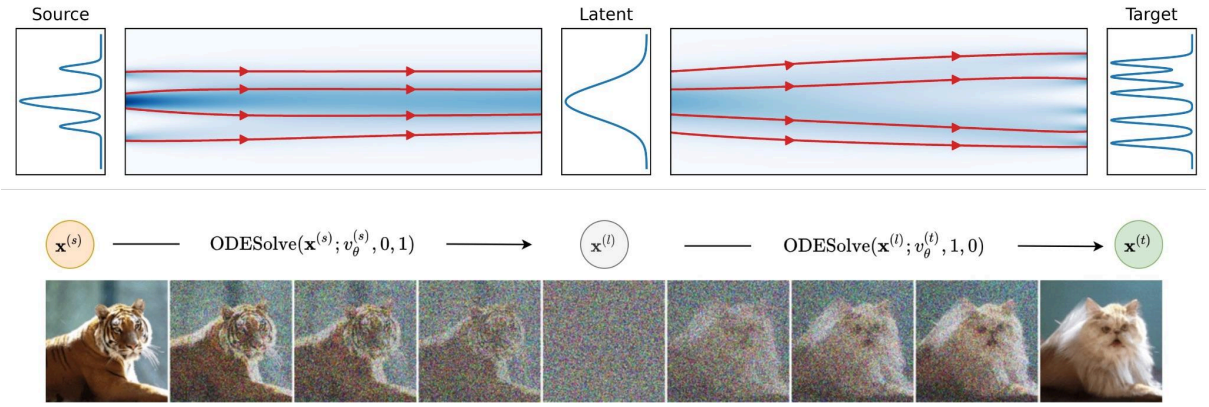
Main goal of the Image to Image translation task is to provide a mapping from one image domain to another. This, can be done in two modes;

- Paired translation: explicit transition from one object to another.
- Unpaired translation: we only have distributions of data, no explicit pairwise relation.

The unpaired case is the most common one and can be solved, for example, with Generative Adversarial Networks or Normalizing Flows. However, the drawback here is the fact that for training these models require access to both the source and target data. In this project we look at this problem through diffusion models: we can train two models to generate our datasets separately and then translate images through the latent codes. The challenge here is to preserve the style of the original image. To tackle it we use Dual Diffusion Implicit Bridge model [1], which uses deterministic stochastic processes in both forward and backward processes. We conducted experiments in space of latent codes and attempted to understand the influence of perturbations on the final result.

Related Work

Image-to-Image translation task might be solved by diffusion models, where the encoder can be represented by the first DDPM and the decoder by the second one. Specifically, the encoder can provide us the latent representation of the object from the first domain, containing some stylistic features, while the decoder will sample the object from the second domain, based on this latent noise. This idea was developed in Dual Diffusion Implicit Bridges for Image-to-Image Translation (DDIB) [1]. In DDIB authors leverage two ODE solvers for image translation: one runs ODE in forward to convert it to latent and the second one goes in reverse mode, constructing the target image from the latent:



Methodology

The central part of DDIB is the probability flow - it is a deterministic ODE that carries the same marginal densities as the diffusion process throughout its trajectory. Thus, when we execute either forward or backward process in DDIB we actually solve the probability flow ODE:

$$dx = [f(x,t) - \frac{1}{2}g(t)^2 \nabla_x \log p_t(x)]dt.$$

To approximate the score function $\nabla_x \log p_t(x)$ we use θ-parameterized score network $s_{t,\theta} \approx \nabla_x \log p_t(x)$. Training is done all thanks to a variational lower bound.

Now authors suggest that we may solve this ODE with the help of ODE solvers, after replacing the score function with our approximation network, which allows us to compute x at each time step:

$$\text{ODESolve}\,(x(t_0);\, v_\theta\,,\, t_0,\, t_1) = x(t_0) + \int_{t_0}^{t_1} v_\theta(t,\, x(t))dt,$$

where $v_\theta(t,\, x(t)) = \frac{dx}{dt}$. Therefore, we observe a deterministic process and can understand the high-level idea of DDIB:

High-level Algorithm DDIB

Input: data sample from source domain $x^{(s)} \sim p_s(x)$, source model $v_\theta^{(s)}$, target model $v_\theta^{(t)}$.

Output: $x^{(t)}$, the result in the target domain.

1. $x^{(l)} =$ ODESolve $(x^{(s)};\, v_\theta^{(s)},\, 0,\, 1)$ - obtain latent code from source domain data
2. $x^{(t)} =$ ODESolve $(x^{(l)};\, v_\theta^{(t)},\, 1,\, 0)$ - obtain target domain data from latent code

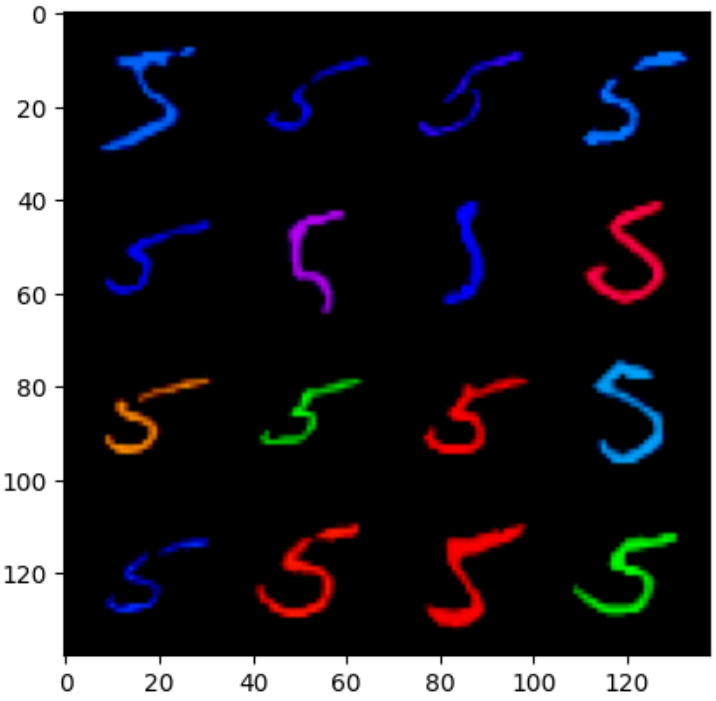
---

In our experiments we constructed a colored MNIST dataset [2] and trained a DDIB model to generate various versions of digit seven in different colours. Our goal was to apply different perturbations of latent code and to see how they may affect the result. During these experiments we attempted to generate sevens from fives, conserving the colour of the digits. We did that by applying gaussian blur and rotation operations.

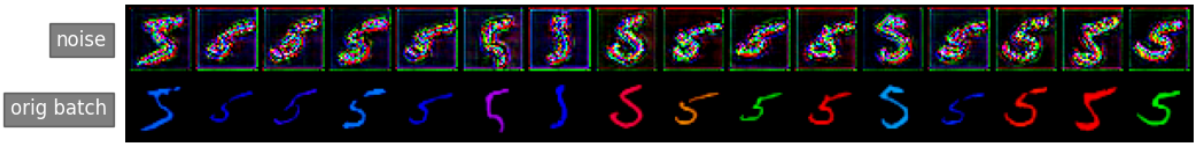
Results

All images with description

We trained a DDPM model capable of generating colored digits seven.
Like initial batch (or source) we take colored digits five:

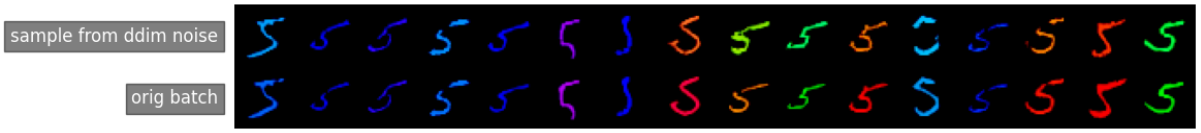


Then noise this batch with ddim.reverse_sample_loop:

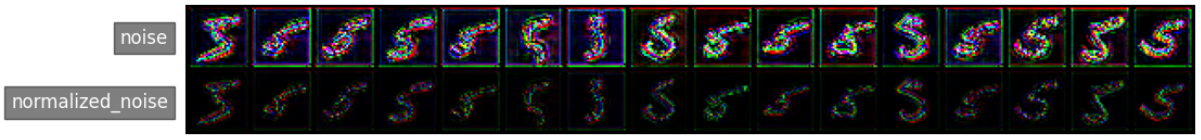


Main experiments

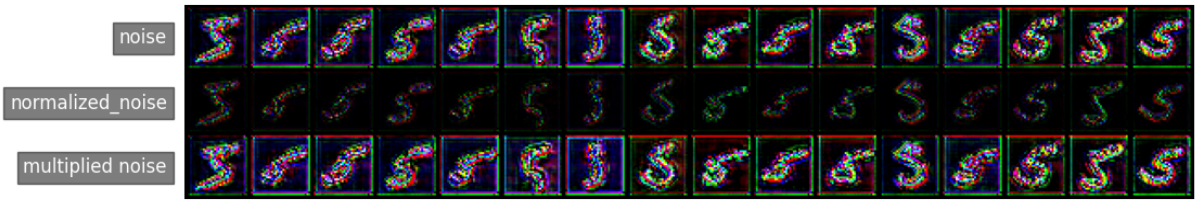
1. Try to generate digits 7 from this latent noise



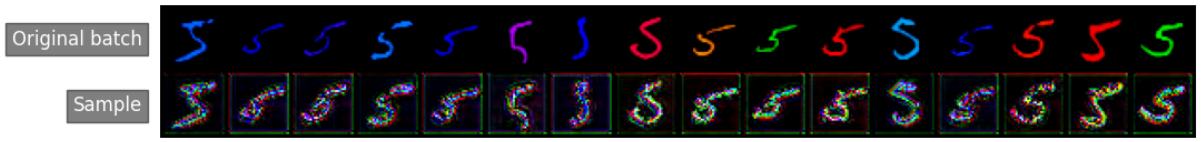
2. Normalize noise:



Sampling from normalized was bad (almost black result).
Multiply by some coefficient:



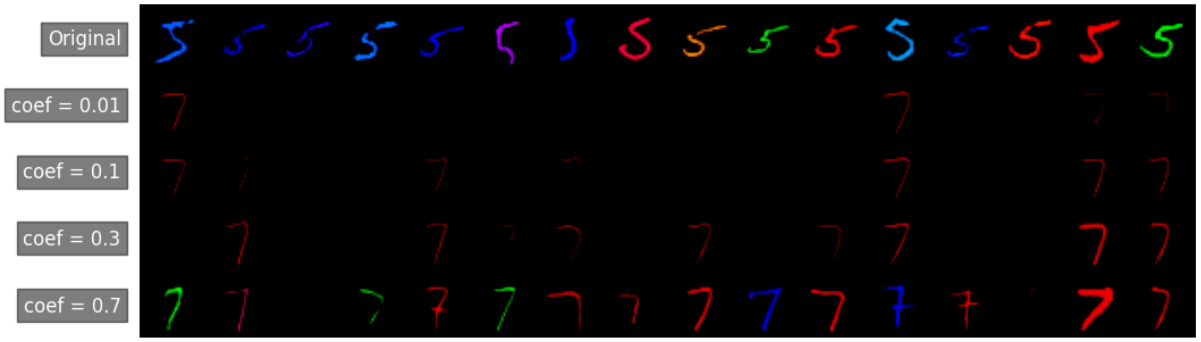
And sample:



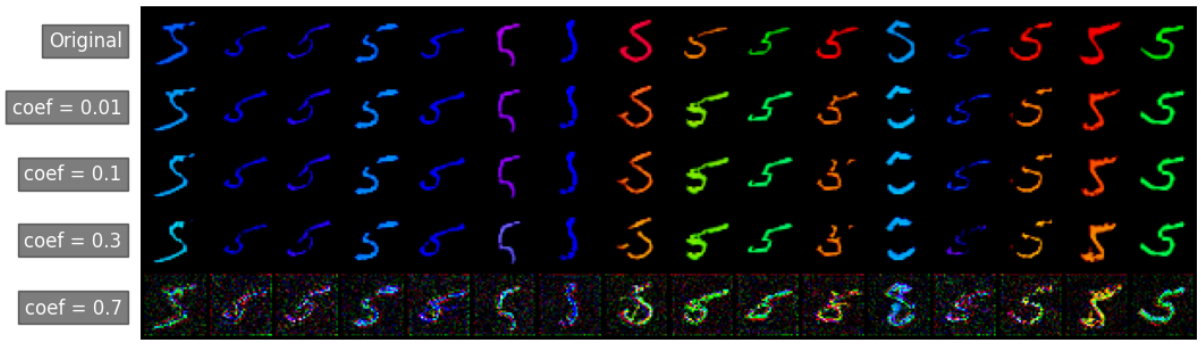
Results got worse

3. Try to add some fixed Gauss noise to with some coefficient to:

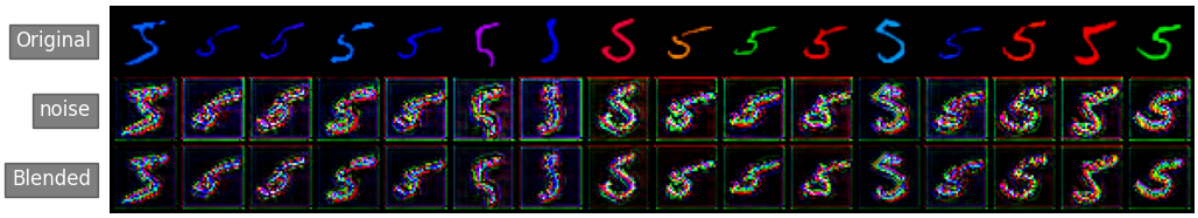
a. `noise=normalized_noise + c * fixed_noise` and iterate over `coefs = [0.01, 0.1, 0.3, 0.7]`



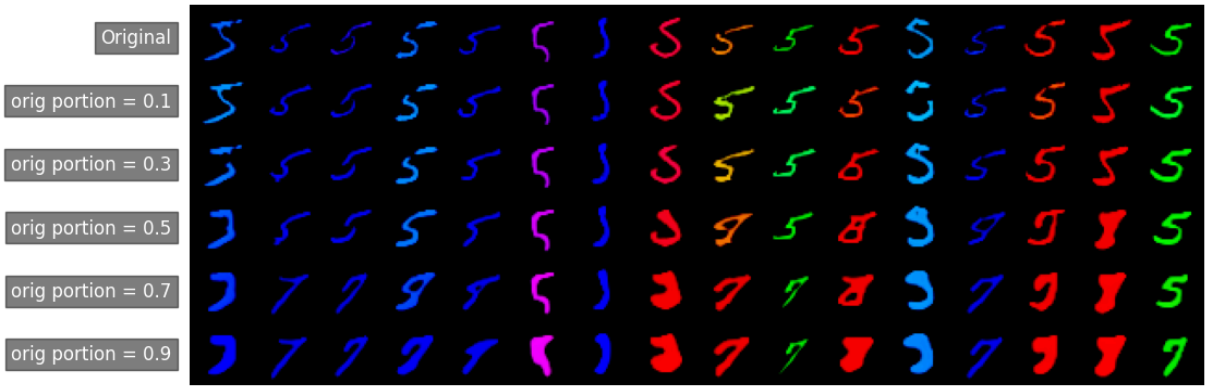
b. `noise=noise + c * fixed_noise .` (source noise)
and iterate over `coefs = [0.01, 0.1, 0.3, 0.7]`



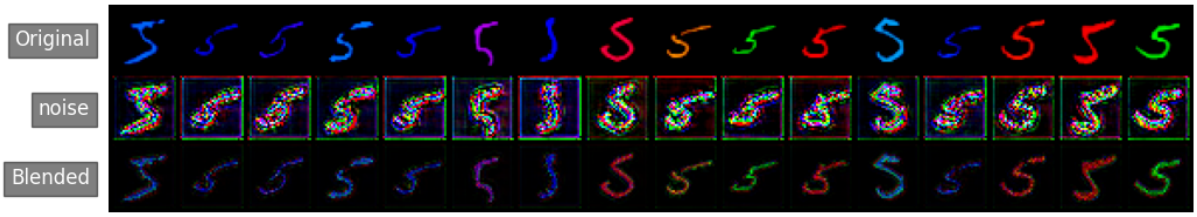
4. Blending original image with its latent with different proportions:
`proportions = [0.1, 0.3, 0.5, 0.7, 0.9]`
    a. source noise



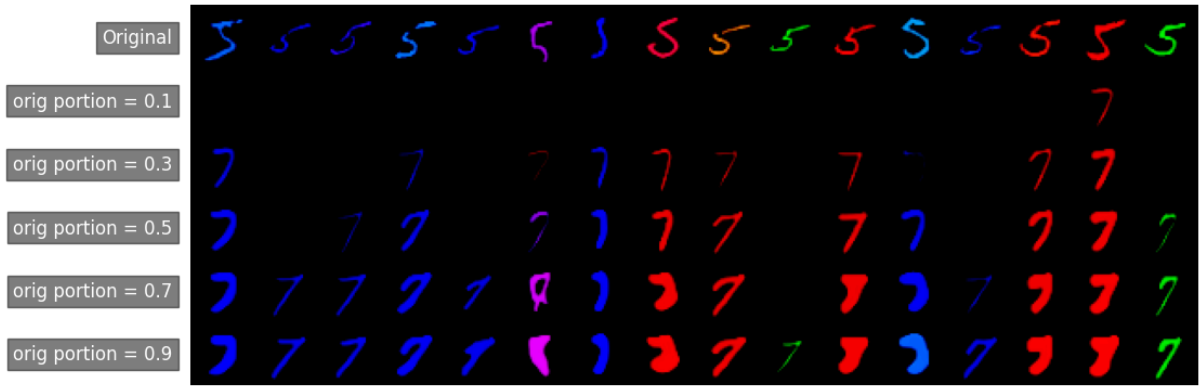
Result:



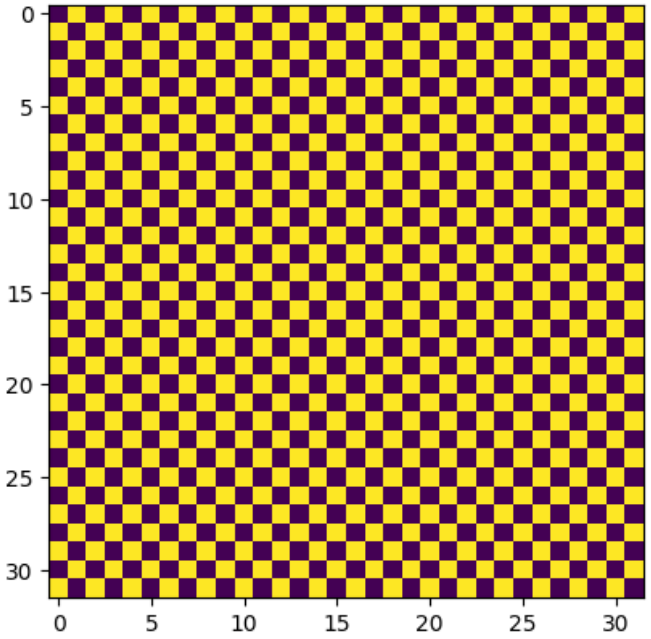
    b. normalized noise



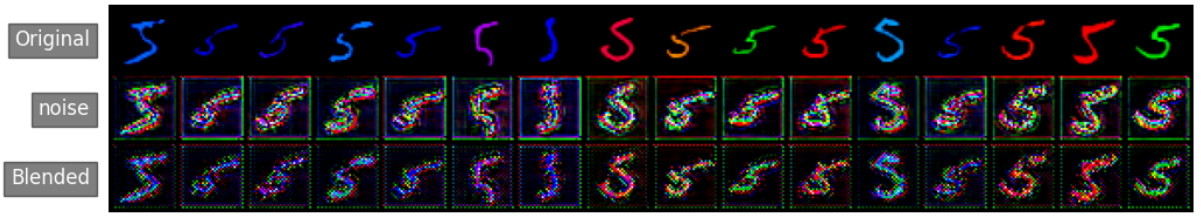
Result:



5. Blending in checkerboard style with original image

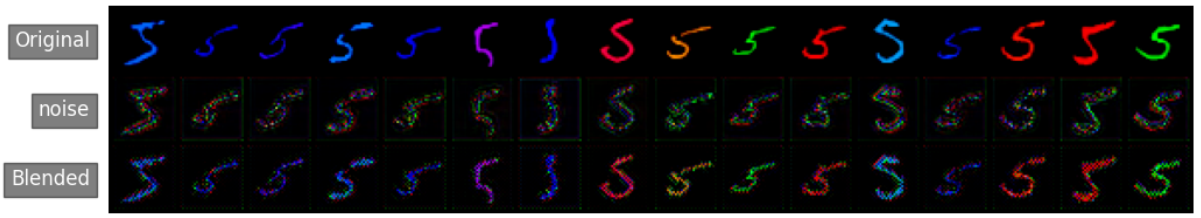


a. Source noise

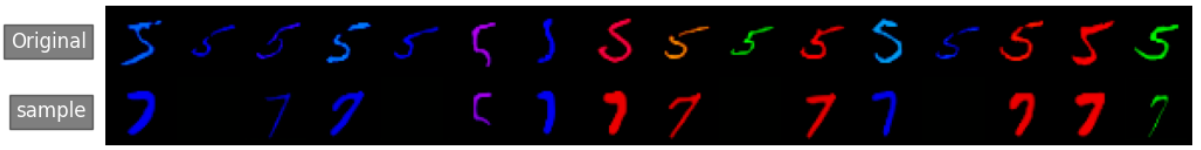


Result:

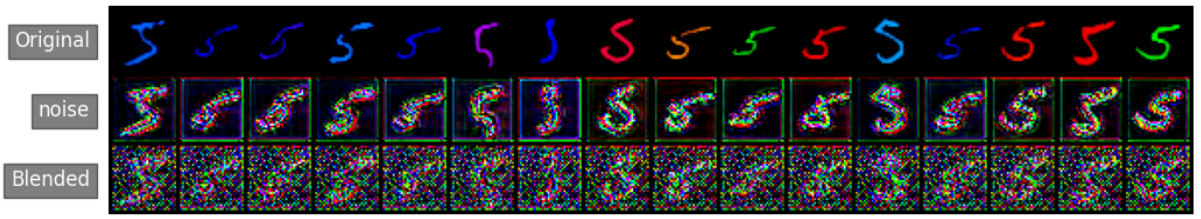
b. Normalized noise



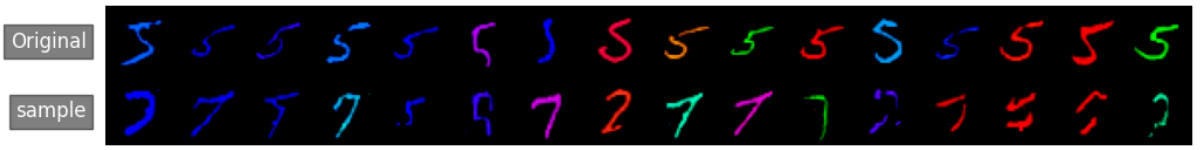
Result:



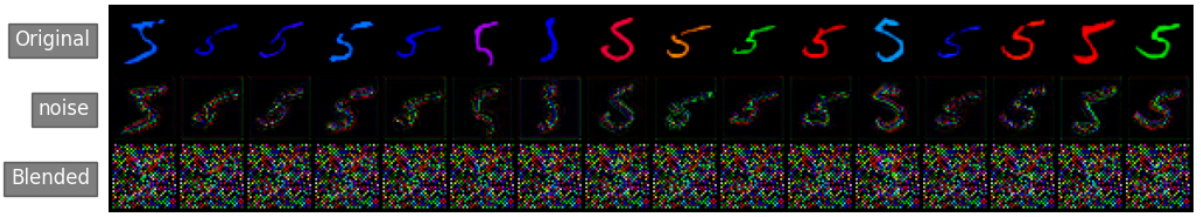
6. Blending in checkerboard style with fixed Gaussian nose
   a. Source noise



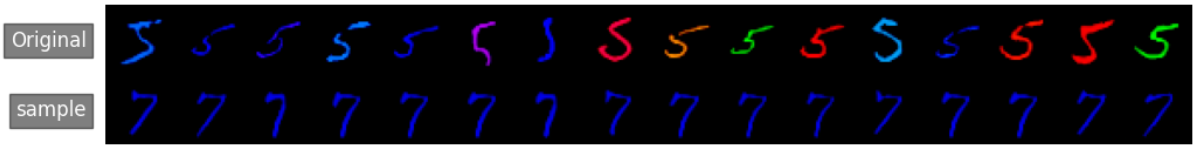
Result:



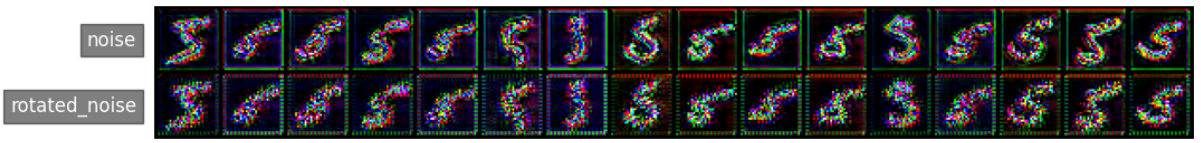
   b. Normalized noise



Result:

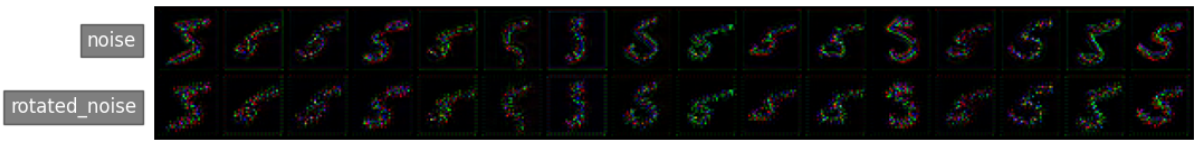


7. Small local rotation. Split latent channel-wise, every channel split on 2x2 areas. Assume an arbitrary small 2x2 area, for example: [[a, b], [c, d]]. Then apply a clockwise rotation:: [[c, a], [d, b]].
   a. Source nose



Result:

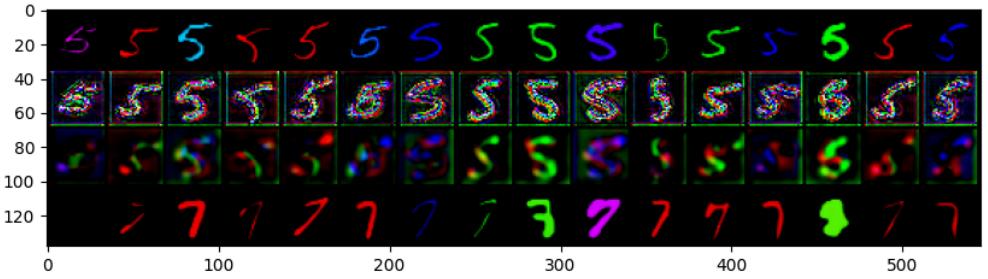
b. Normalized noise



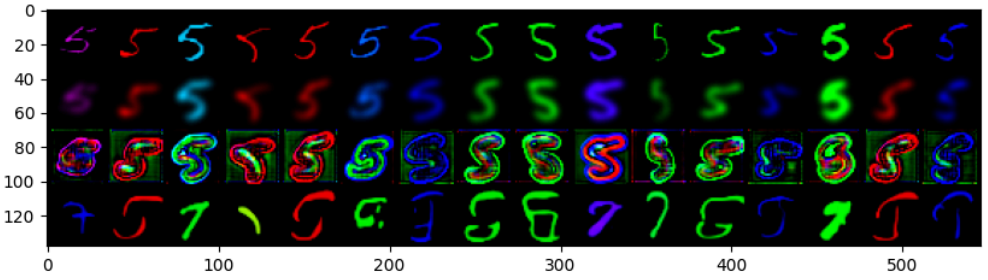
Result:

Additional experiments

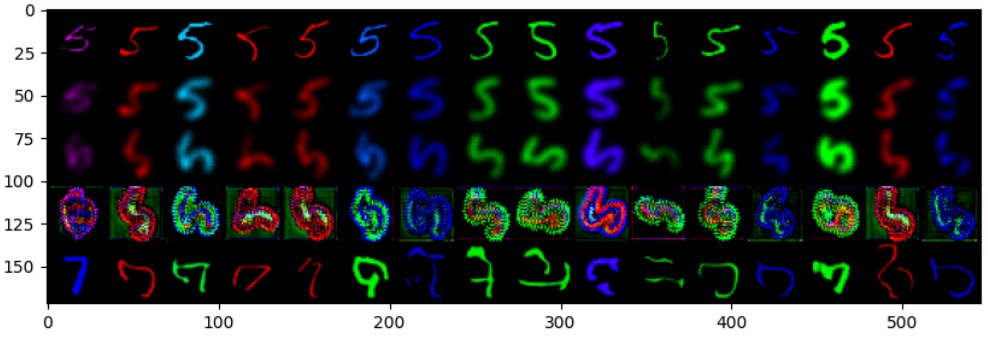
First, the idea was to apply blur to the original image and see how it will go:



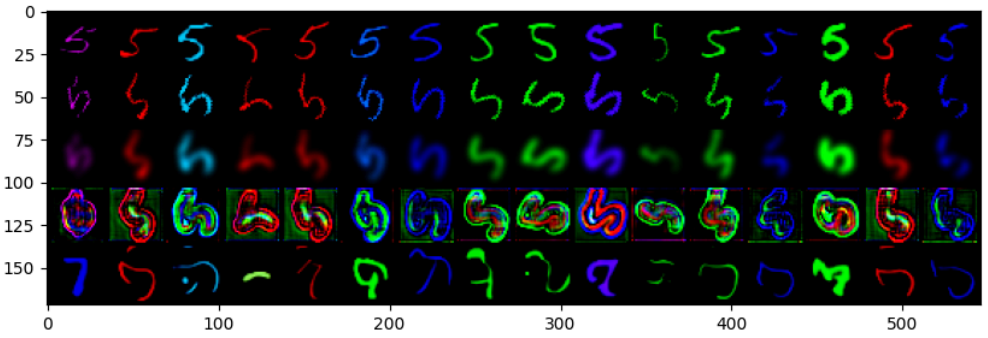
Blur almost removed the structure of the original digit, however it also altered the color. Then we switched places and applied gaussian blur to the noise itself:



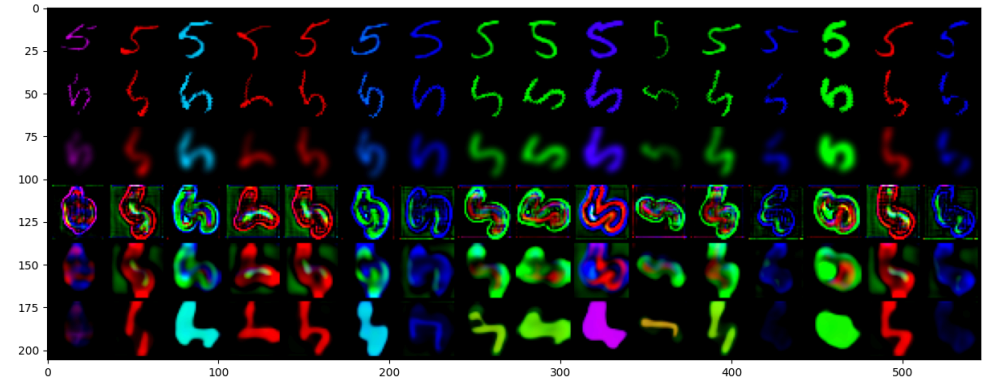
Here we faced an interesting artefact: latent code gives a colorful contour to the digit. Now add rotation to the latent:



Another interesting observation: the model is sensitive to the artefacts, which rotation creates. Now let's rotate and then blur:



The received latent is almost close to what we've got without rotation, thus we have slight equivariance. After that we decided to blur the latent itself once more:



Well, we got the colors almost, but failed to get the needed structure of seven.

Conclusion

In this project we trained a DDIB model to generate sevens from the colored MNIST dataset we created. We conducted experiments by getting the latent code of one digit and attempting to convert into seven, while preserving the colour. We applied blur, rotation and noise to the latent code, and observed some interesting phenomena in the latent space. The best performance so far was obtained, when we applied noise in checkerboard fashion, which shows that we managed to eliminate the structure of the original image and preserve its core feature. The code supporting the results can be found on our github.

References

1. Xuan Su, Jiaming Song, Chenlin Meng, Stefano Ermon, “Dual Diffusion Implicit Bridges for Image-to-Image Translation”, 2023, ICLR’23
2. Nikita Gushchin, Alexander Kolesov, Alexander Korotin, Dmitry P. Vetrov, Evgeny Burnaev, “Entropic Neural Optimal Transport via Diffusion Processes”, NeurIPS’23