Internal Learning with Diffusion Model

Haojun Qiu
(joint work with Wenzheng Chen, David Lindell, Kyros Kutulakos)

- Hi, I’m going to present my project, the title I gave here is "internal learning with DM" (and this is advised by David & Kyros & Wenzheng)

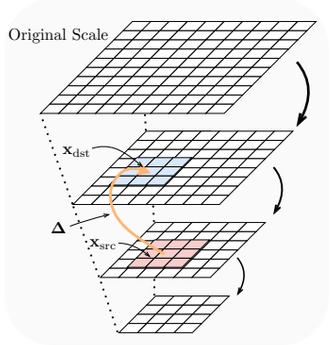
1. Given a single image

- Let’s first see the whole pipeline, they will all be addressed in more details later
- First, we are given (and only given) a single image in the wild

1. Given a single image

---

2. Extract patch level dataset/joint distribution



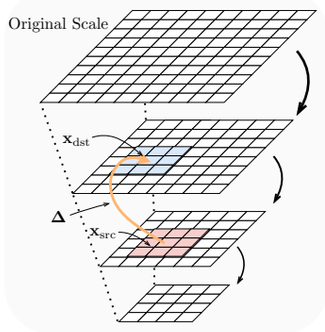
$$\mathbf{\Delta} = \begin{bmatrix} \theta_{\text{src}} \\ u \\ v \\ s_x \\ s_y \end{bmatrix} \begin{matrix} \} \text{ rotation degree} \\ \} \text{ displacements} \\ \} \text{ scale} \end{matrix}$$

$$\{(\mathbf{x}_{\text{src}}, \mathbf{\Delta}, \mathbf{x}_{\text{dst}})^{(i)}\}_{i=1}^{N}$$

- You can construct multi-scale representation and extract some patch level data from it
- [Basically, two neighboring patches from the image pyramid as well as their relative spatial relationship]

1. Given a single image

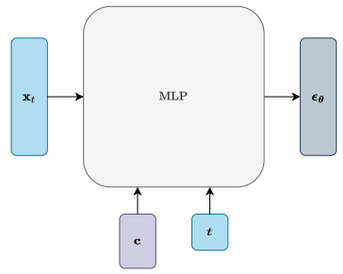
2. Extract patch level dataset/joint distribution



$$\boldsymbol{\Delta} = \begin{bmatrix} \theta_{\text{src}} \\ u \\ v \\ s_x \\ s_y \end{bmatrix} \begin{matrix} \} \text{ rotation degree} \\ \} \text{ displacements} \\ \} \text{ scale} \end{matrix}$$

$$\{(\mathbf{x}_{\text{src}}, \boldsymbol{\Delta}, \mathbf{x}_{\text{dst}})^{(i)}\}_{i=1}^{N}$$

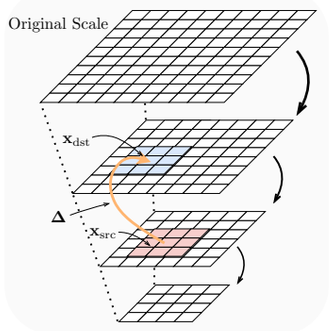
3. Train patch level diffusion models — conditional/unconditional



- Then, we’re going to train some diffusion models from scratch to learn patch level distribution

1. Given a single image

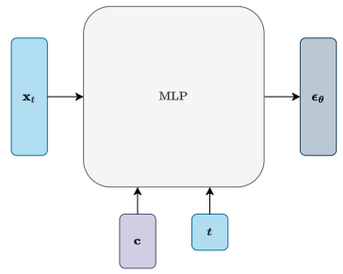
2. Extract patch level dataset/joint distribution



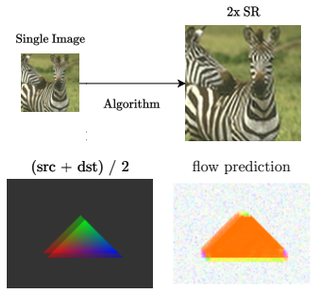
$$\boldsymbol{\Delta} = \begin{bmatrix} \theta_{\text{src}} \\ u \\ v \\ s_x \\ s_y \end{bmatrix} \begin{matrix} \} \text{ rotation degree} \\ \} \text{ displacements} \\ \} \text{ scale} \end{matrix}$$

$$\{(\mathbf{x}_{\text{src}}, \boldsymbol{\Delta}, \mathbf{x}_{\text{dst}})^{(i)}\}_{i=1}^{N}$$

3. Train patch level diffusion models — conditional/unconditional



4. Applications like Super-Resolution, Optical Flow, etc.



- Lastly, the trained model can be used for different tasks on the same image given, for example, super-resolve the given image, or estimate the flow given a new neighboring frame

Overview

- Objectives (what we do)
- Significance (why we do it)

Two things I wanna make clear upfront are: (1) what we're doing and (2) why we're doing it.

Overview

Objectives (what we do):

- **Internal Learning with Diffusion Models.** Learning *patch-level* distributions (conditional, unconditional) from a single image, by training diffusion models from scratch.

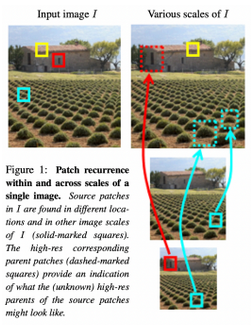
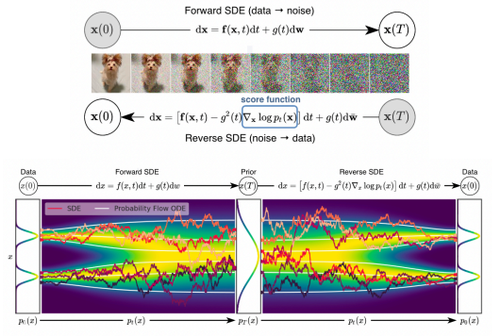


Figure 1: **Patch recurrence within and across scales of a single image.** *Source patches in I are found in different locations and in other image scales of I (solid-marked squares). The high-res corresponding parent patches (dashed-marked squares) provide an indication of what the (unknown) high-res parents of the source patches might look like.*



I think for what we are doing — it’s kinda repeating the title, which involves two domains
(1) the problem domain is the 'internal learning', which is about modeling distribution of patches / exploit statistics of patches from a single image (Michal Irani’s group has been working on this topic for a long time)
(2) then the tool we choose to model the distributions is the diffusion model

Overview

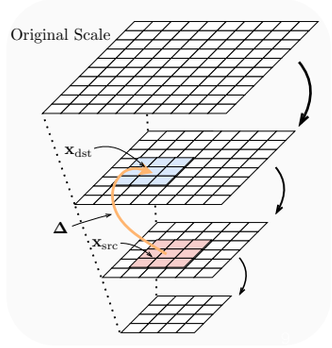
- Objectives (what we do)
- Significance (why we do it)

- There has been some works on combining these two fields — for example famous SinGAN and its following works.

Overview

Significances (Why we do it?):

- We make a difference by analyzing patches joint distributions *p*(patch1, patch2, relationship)



$$\boldsymbol{\Delta} = \begin{bmatrix} \theta_{\text{src}} \\ u \\ v \\ s_x \\ s_y \end{bmatrix} \begin{matrix} \} \text{ rotation degree} \\ \} \text{ displacements} \\ \} \text{ scale} \end{matrix}$$

$$\{(\mathbf{x}_{\text{src}}, \boldsymbol{\Delta}, \mathbf{x}_{\text{dst}})^{(i)}\}_{i=1}^{N}$$

- The significance of ours is in modeling this ’joint distribution’ in the red box (two patches plus their relationship)
- We believe this more general setting allows you do some different tasks that the prior works cannot do

Our Approach

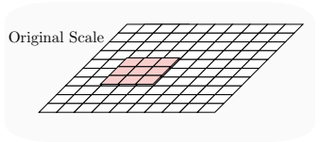
- Data representation (what is the distribution)
- Adapt diffusion model for our data representation

- Now I’m going to walk you through what’s the joint distribution and where it comes from

Our Approach — Data Representation

Distribution of patch

- Given a single image (in experiment mostly use sized 250 x 250), we crop small patches 7 x 7

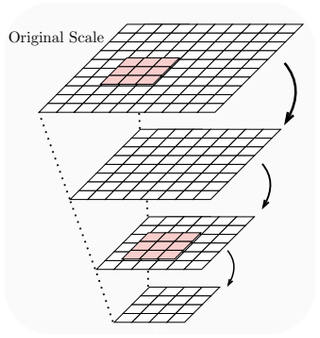


- Given a single image, we consider set of small patches, typically 7 by 7, and each is treated as an individual data point (for visualization I draw the whole thing in sparse grids)

Our Approach — Data Representation

Distribution of patch

- Over-scales too

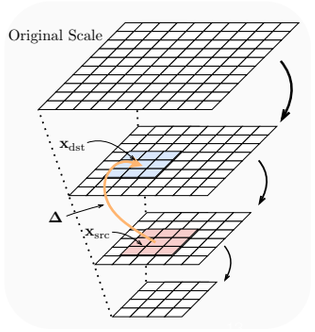


- You can construct a image pyramid and extract the same-sized (7x7) patch at any scale
- The idea is: in lower scales the patch represents more abstract information (like object shape or layout) and in finer resolution it is about, that’s say, textures

Our Approach — Data Representation

Joint distribution of (patch1, patch2, relationship)

- **A core difference** from prior work is the joint distribution of two patches and their 'relationship'.



$$\boldsymbol{\Delta} = \begin{bmatrix} \theta_{\text{src}} \\ u \\ v \\ s_x \\ s_y \end{bmatrix} \begin{matrix} \} \text{ rotation degree} \\ \} \text{ displacements} \\ \} \text{ scale} \end{matrix}$$

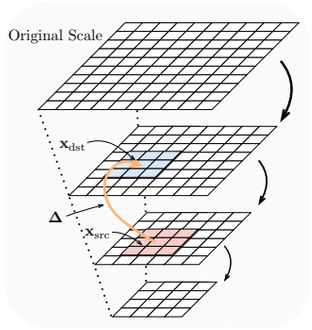
$$(\mathbf{x}_{\text{src}}, \boldsymbol{\Delta}, \mathbf{x}_{\text{dst}})$$

- Okay now, the core difference comes from this joint distribution
- You not only can get individual patch samples, but you can retrieve any two neighboring patches from the image pyramid, as well as their relative spatial information we called this delta (which contains their relative displacement and relative scales. And also, they are small. For example, you can imagine the relative displacement is, by construction, set to be smaller than the patch size)

Our Approach — Data Representation

Joint distribution of (patch1, patch2, relationship)

- What makes **a core difference** from prior work is the joint distribution of two patches and their 'relationship'.



$$\boldsymbol{\Delta} = \begin{bmatrix} \theta_{\text{src}} \\ u \\ v \\ s_x \\ s_y \end{bmatrix} \begin{matrix} \} \text{ rotation degree} \\ \} \text{ displacements} \\ \} \text{ scale} \end{matrix}$$

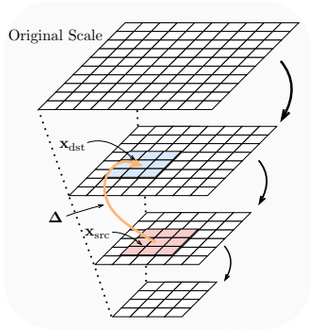
$$\left(\mathbf{x}_{\text{src}}, \boldsymbol{\Delta}, \mathbf{x}_{\text{dst}}\right)$$

- So essentially this dataset of (local) triplet extracted from the single image is what we deal with
- And this joint distribution on its own can be interesting but actually it borns two types of (perhaps even more) interesting conditional distributions

Our Approach — Data Representation

Joint distribution of (patch1, patch2, relationship)

- What makes **a core difference** from prior work is the joint distribution of two patches and their 'relationship'.



$$\boldsymbol{\Delta} = \begin{bmatrix} \theta_{\text{src}} \\ u \\ v \\ s_x \\ s_y \end{bmatrix} \begin{matrix} \} \text{ rotation degree} \\ \} \text{ displacements} \\ \} \text{ scale} \end{matrix}$$

$$(\mathbf{x}_{\text{src}}, \boldsymbol{\Delta}, \mathbf{x}_{\text{dst}})$$

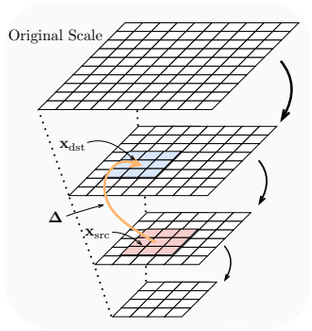
$$(\mathbf{x}_{\text{src}}, \boldsymbol{\Delta}) \mapsto \mathbf{x}_{\text{dst}} \quad p(\mathbf{x}_{\text{dst}} \mid \mathbf{x}_{\text{src}}, \boldsymbol{\Delta})$$

- One is this patch prediction direction, given a source patch and 'delta', you would like to sample a distribution of patches you end up getting, here so-called the destination (dst) patch
- Thinking about in SR: x_src is LR patch, and delta is a vector stores "hey, let's scale up by a factor of 2", then there should be a set of samples of HR patches as x_dst.
- Note that we have this idea of conditional distribution (rather than deterministic mapping) b/c of patch recurrence (basically you have one to many correspondence in your dataset)

Our Approach — Data Representation

Joint distribution of (patch1, patch2, relationship)

- What makes **a core difference** from prior work is the joint distribution of two patches and their 'relationship'.



$$\boldsymbol{\Delta} = \begin{bmatrix} \theta_{\text{src}} \\ u \\ v \\ s_x \\ s_y \end{bmatrix} \begin{matrix} \} \text{ rotation degree} \\ \} \text{ displacements} \\ \} \text{ scale} \end{matrix}$$

$$(\mathbf{x}_{\text{src}}, \boldsymbol{\Delta}, \mathbf{x}_{\text{dst}})$$

$$(\mathbf{x}_{\text{src}}, \boldsymbol{\Delta}) \mapsto \mathbf{x}_{\text{dst}} \quad p(\mathbf{x}_{\text{dst}} \mid \mathbf{x}_{\text{src}}, \boldsymbol{\Delta})$$

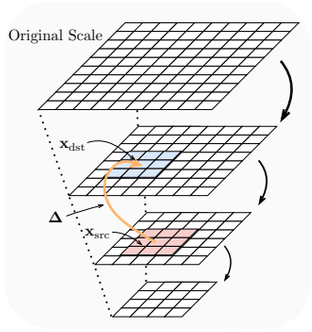
$$(\mathbf{x}_{\text{src}}, \mathbf{x}_{\text{dst}}) \mapsto \boldsymbol{\Delta} \quad p(\boldsymbol{\Delta} \mid \mathbf{x}_{\text{src}}, \mathbf{x}_{\text{dst}})$$

- Another interesting direction is, this, mapping two patches to their relationship delta. For example given two patches can you estimate what's their displacements apart from each other?
- And again, it's a conditional distribution for the similar reason (ambiguity)

Our Approach — Data Representation

Joint distribution of (patch1, patch2, relationship)

- What makes **a core difference** from prior work is the joint distribution of two patches and their 'relationship'.



$$\boldsymbol{\Delta} = \begin{bmatrix} \theta_{\text{src}} \\ u \\ v \\ s_x \\ s_y \end{bmatrix} \begin{matrix} \} \text{ rotation degree} \\ \} \text{ displacements} \\ \} \text{ scale} \end{matrix}$$

$$\begin{aligned} &(\mathbf{x}_{\text{src}}, \boldsymbol{\Delta}, \mathbf{x}_{\text{dst}}) \\ &(\mathbf{x}_{\text{src}}, \boldsymbol{\Delta}) \mapsto \mathbf{x}_{\text{dst}} \quad p(\mathbf{x}_{\text{dst}} \mid \mathbf{x}_{\text{src}}, \boldsymbol{\Delta}) \\ &(\mathbf{x}_{\text{src}}, \mathbf{x}_{\text{dst}}) \mapsto \boldsymbol{\Delta} \quad p(\boldsymbol{\Delta} \mid \mathbf{x}_{\text{src}}, \mathbf{x}_{\text{dst}}) \end{aligned}$$

- Okay, that’s about the data representation

Our Approach

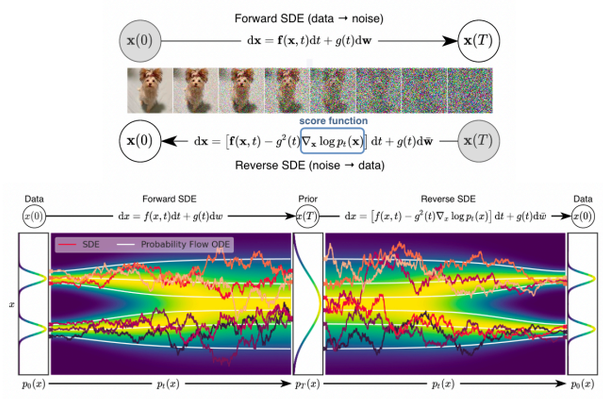
- Data representation (what is the distribution)
- Adapt diffusion model for our data representation

- Now we move to adapting diffusion models for our use-case, and this is actually a simpler part to think about.

Our Approach — Diffusion Model on Vectors

Mathematical framework

- Choose one of the framework out-of-the-box, DDPM, SDE, ...



- Typically, the design of diffusion model consist of two things, the mathematical framework (e.g., forward/reverse processes) + a neural network score/denoising function
- For mathematical framework, we just choose one out-of-the-box, for example, DDPM, SDE, and etc.

Our Approach — Diffusion Model on Vectors

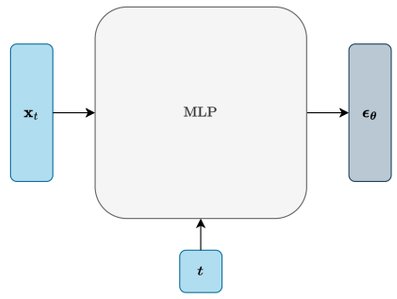
Change the denoiser

- U-net is for big 2D images.
- Given our data are mostly 'vector-like' (small patch is flattened as a vector), we just use MLP for denoiser

- The thing we need to re-design is just the denoiser.
- Most works are about learning on large images, so they use 2D U-net as denoiser
- But given our data are all vector-like (for example, the patch is small enough you want to flatten it to a vector, delta is also a vector), we just opt for a MLP

Our Approach — Diffusion Model on Vectors

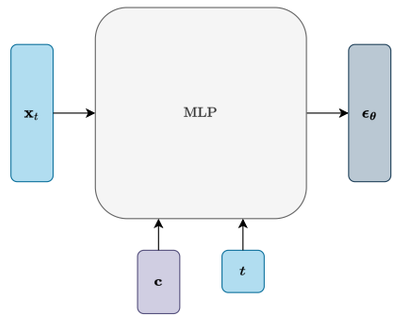
Unconditional model — $p(\mathbf{x})$



- This is what it looks like for unconditional model/denoiser
- The only restriction for the MLP is to have same input and output dimensions, and have a time conditioning as input

Our Approach — Diffusion Model on Vectors

Conditional model — $p(\mathbf{x} \mid \mathbf{c})$



- for conditional model though, you need to input a extra conditioning signal **c** in the reverse denoising process
- The two conditional distributions mentioned before are just like this, except that you want to replace data x, and conditioning c for different things in each case

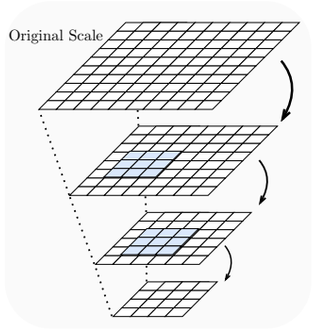
Results

Unconditional Generation

- Now let’s first look at some experiments on unconditonal patch generation.
- Given this single image here I’m gonna learn a unconditional patch distribution from different (and all) layers of the pyramid

Results

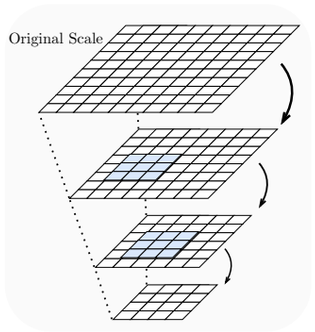
Unconditional Generation

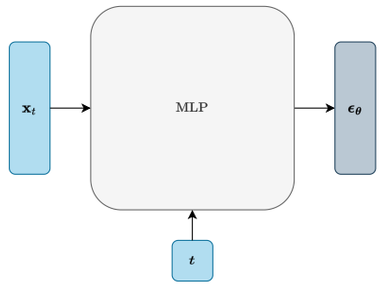


- So we first construct the pyramid
- Then cropping out small patches as a dataset (here it’s about unconditional model so it’s not a triplet yet, each data is just the patch itself)

Results

Unconditional Generation





- Then, the denoiser learns to denoise the pertubed version of patches at many noise level, just like all these image diffusion models

Results

Unconditional Generation

Generated

- After training, here are some randomly generated patches

Results

Unconditional Generation

Generated

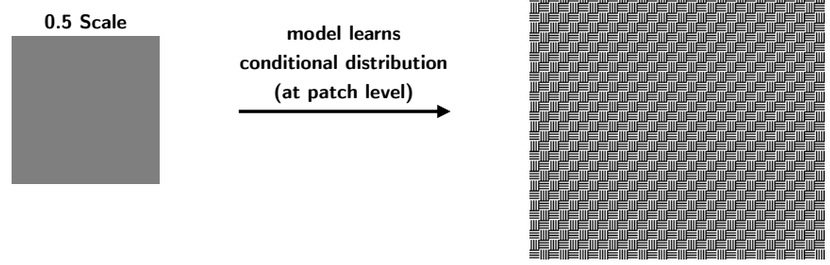
Real

- As a reference, on the right panel, I randomly sampled some real patches
- The two look like coming from the *same* distribution, that's what we wanted.
- And of course, we've got some quantitative metrics to evaluate it, but I would not be able to cover that here

Results

Conditional generation (patch) — synthetic data

$$\boxed{p(\mathbf{x}_{\text{dst}} \mid \mathbf{x}_{\text{src}}, \boldsymbol{\Delta})}$$

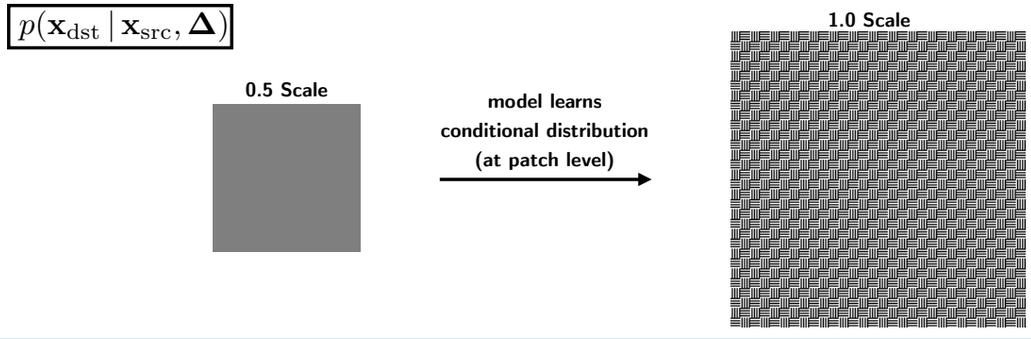


- Now we get to see some experiments on conditional distribution of patches (this is about mapping (a patch + delta) to another patch)
- It will be a synthetic data here for just proof of concept
- We want the model to learn the mapping from patch in 0.5 scale to 1.0 (original) scale of the image
- The 1.0 scale image on the right is constructed such that when you downsample it to its half-size you get a uniform, non-texture grey image (on the left)
- So there is obviously a **one to many** correspondences (or, ambiguity) from left to right, by construction

Results

Conditional generation (patch) — synthetic data

$$p(\mathbf{x}_{\text{dst}} \mid \mathbf{x}_{\text{src}}, \boldsymbol{\Delta})$$

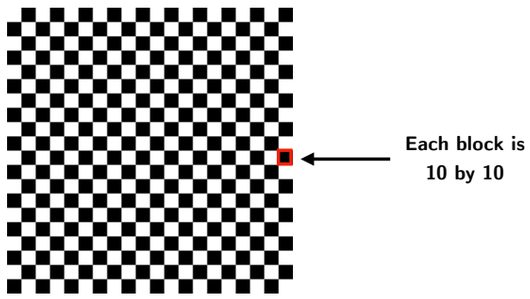


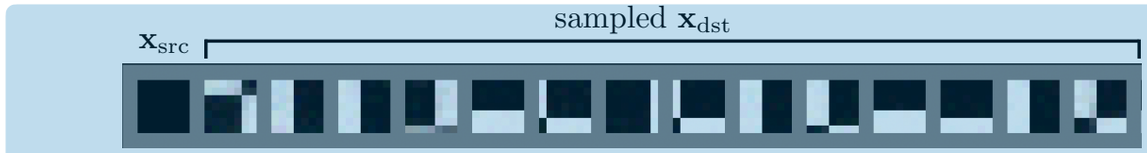
- Here in the bottom, after training, when you sampling the model conditioning on this src patch (this uniform grey patch), you will get a series of samples as its 'HR' textured patches
- So basically if your dataset contains such 'ambiguity', the diffusion model learns to sample from that ambiguity too, this is a desired and kinda expected behavior

Results

Conditional generation (patch) — synthetic data

$p(\mathbf{x}_{\text{dst}} \mid \mathbf{x}_{\text{src}}, \boldsymbol{\Delta})$





$\boldsymbol{\Delta}$ = displace by (3, 3)

- This is yet another synthetic data. Just quickly go over it, it basically shows that you can conditioned on displacements too (it’s not restricted to relative scale/SR) but any data with this type of one to many correspondence

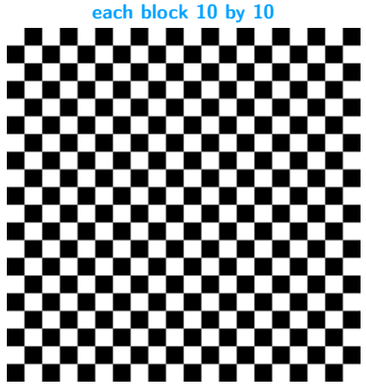
Results

Conditional generation (delta) — synthetic data

$\boxed{p(\boldsymbol{\Delta} \mid \mathbf{x}_{\text{src}}, \mathbf{x}_{\text{dst}})}$

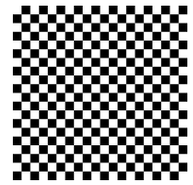
- Checkerboard each block 10 x 10.
- Patch size 7 x 7.
- displacements = $\{-3, -2, -1, 0, 1, 2, 3\}^2$

each block 10 by 10



- Now we consider this mapping from two patches to their relative displacement, as a conditional distribution
- The image used is in checkerboard pattern, each block is 10 x 10 pixels, while the patch used is smaller than the size of the block — as 7 x 7 patch
- We crop neighboring patches with small displacement (within -3 to 3)
- During the training we just feed them into diffusion model as the conditioning signals to generating/denoising their relative displacement

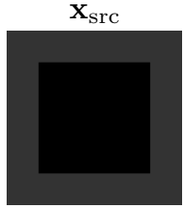
Results



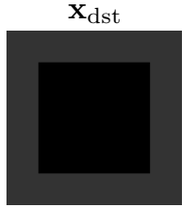
Conditional generation (displacement) — synthetic data

$\boxed{p(\boldsymbol{\Delta} \mid \mathbf{x}_{\text{src}}, \mathbf{x}_{\text{dst}})}$

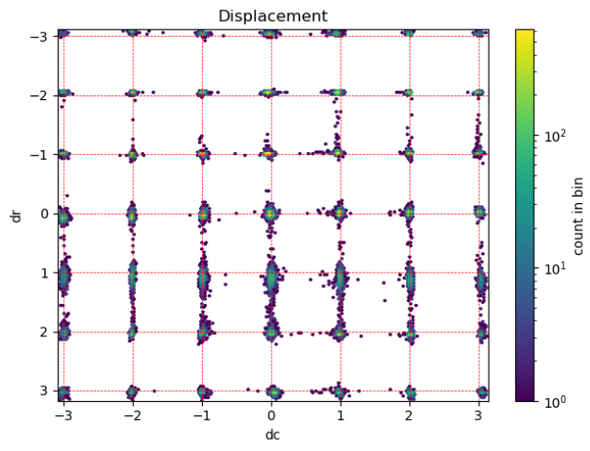
$\mathbf{x}_{\text{src}}$



$\mathbf{x}_{\text{dst}}$

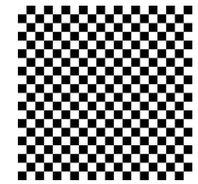


2D Histogram of Generated Displacements



- After training, when you are given two black patches, the samples of displacements appears to be a uniform-like distribution (and that is exactly the type of 1-to-many correspondences in the dataset, b/c this 7x7 patch is within/smaller than one of those 10x10 block in checkerboard image)

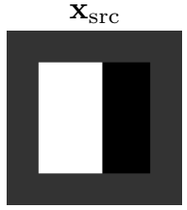
Results



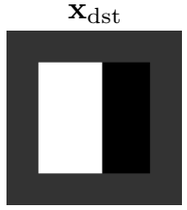
Conditional generation (displacement) — synthetic data

$\boxed{p(\boldsymbol{\Delta} \mid \mathbf{x}_{\mathrm{src}}, \mathbf{x}_{\mathrm{dst}})}$

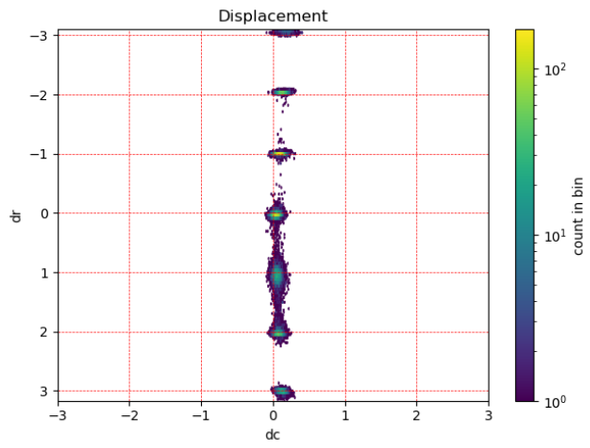
$\mathbf{x}_{\mathrm{src}}$



$\mathbf{x}_{\mathrm{dst}}$

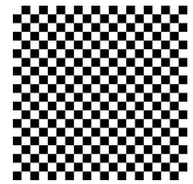


2D Histogram of Generated Displacements



- Here is another pair of patches, for similar reason, you have ambiguity in the vertical direction

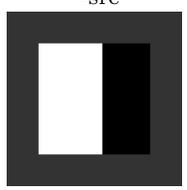
Results



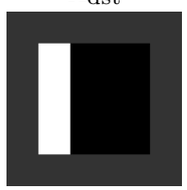
Conditional generation (displacement) — synthetic data

$\boxed{p(\boldsymbol{\Delta} \mid \mathbf{x}_{\mathrm{src}}, \mathbf{x}_{\mathrm{dst}})}$

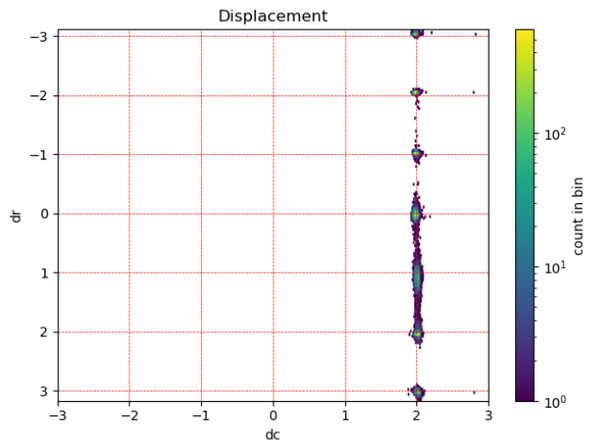
$\mathbf{x}_{\mathrm{src}}$



$\mathbf{x}_{\mathrm{dst}}$



2D Histogram of Generated Displacements

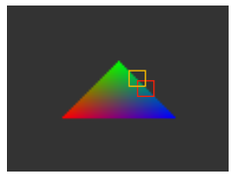


- Here yet another pair of patches
- So this direction of mapping two patches to a displacement seems to work as expected in synthetic data too

Results

Conditional generation (displacement) — synthetic data

training frame



$$p(\underbrace{\boldsymbol{\Delta}}_{\text{disp.}} \mid \mathbf{x}_{\text{src}}, \mathbf{x}_{\text{dst}})$$

- (Check the time, can leave this optical flow till the end)
- Then we make it to predict displacements for across images — you can see the red & yellow crops that's two neighboring patches from the *single image*.
- Still, we want to train the displacements estimation on only a single image/frame (that's say @ time *t*).

Results

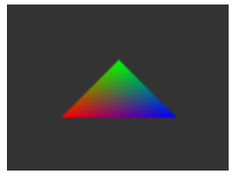
Conditional generation (displacement) — synthetic data

- At inference time, we are given an unseen new frame (let's say @ time *t*+1 or *t*+2).
- This time, to predict 'corresponding displacement' across frames, we feed the patches at same location in two frames into the model (here the red and yellow patch locates at the same position) — and then the model will predicts the displacements between them
- Intuitively the key assumption made here is the patch distribution from frame t and frame t+1 are very similar, so training done on frame t would suffice. At inference time model should generalizes when patches no longer just from the training frame (but the new frame as well)

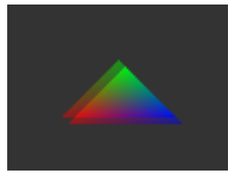
Results

Conditional generation (displacement) — synthetic data

training frame



(train + new) / 2



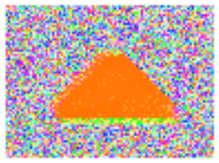
GT flow visualization

- Here in the middle is when you overlap the two frames, you can see there is truly small displacement and I’m not cheating
- Of course this assumption of same patch distribution is met in this synthetic data case since it’s exactly the same object I moved it a bit from frame to frame. But it could be a reasonable assumption for many real frames too

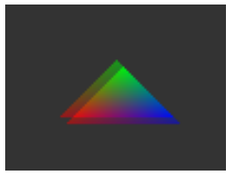
Results

Conditional generation (displacement) — synthetic data

pred (n =1)



(train + new) / 2



GT flow visualization

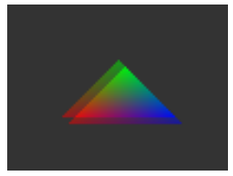
- Here is the result for one displacement sample for patch pairs at every pixel location (when I say patch pair, one from training frame, one from new frame)
- You can see the background is a bit noisy (b/c that’s textureless), the textured triangle looks good though

Results

Conditional generation (displacement) — synthetic data

mean, n = 10

(train + new) / 2



GT flow visualization

- If you take more samples and do the average, you gonna reduce some variance
-

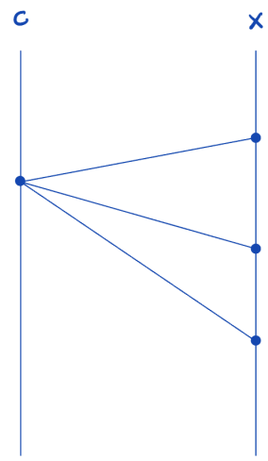
Discovery for real image

Two problems

- (input space) 'recurrence'/generalization
- (output space) memorization
- both occurs —> memorizing pairs

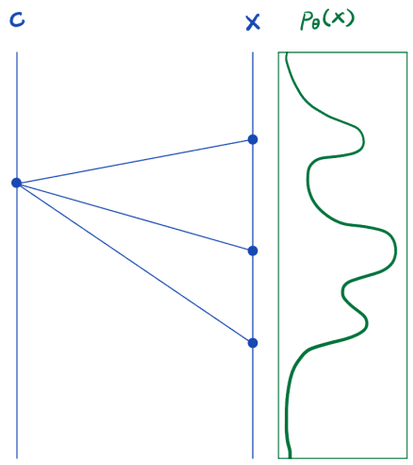
- So that was proof-of-concept for conditional distribution modeling, they works as expected on the synthetic data.
- Then, starting from here its more about findings when we trying to transfer to real image
- There are two issues we found in experiment, and during the process of dealing with them we found an interesting equivalence of two classes of methods (classical search-based, deep generative model)
- At a high level, one of the problem is in the input space (recurrence), one in the output space (memorization/generation by retrieval)
- I will demonstrate them with some simple 1d illustration

Recurrence & Memorization



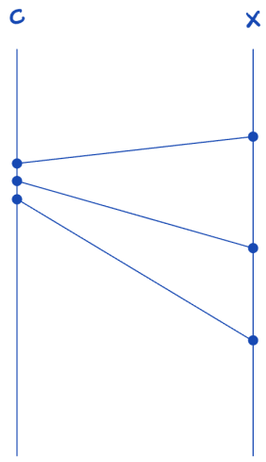
- Previously the synthetic data I’ve shown you trying to map from exact one input (conditioning) to multiple generation samples
- That is, the idea of recurrence is "real/exact recurrence" and this is exactly a 1-to-many correspondence in your training data

Recurrence & Memorization



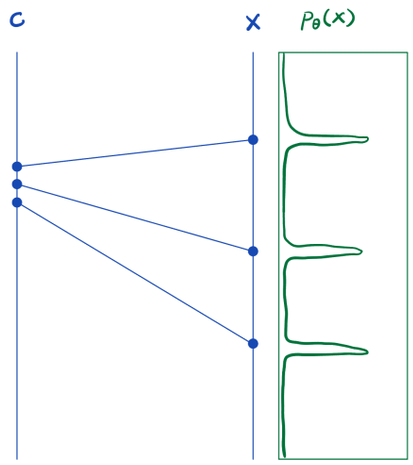
- And after we train a conditional diffusion model, ideally you can sample from some distribution like this (smooth, and contains novel samples, but the modes still locate at the few supervision signals)

Recurrence & Memorization



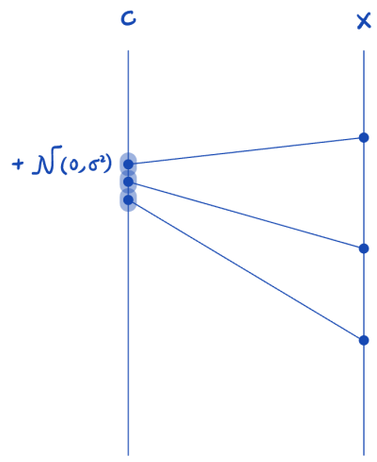
- In real image data though, input small patches rarely recurs/duplicates perfectly. Here for example three inputs **c** are similar that you want to kinda treat them the same — the idea of patch search/match (but they still exist as distinct points)
- So this is no longer like before where you have one to many correspondence, it’s a bijective correspondence here

Recurrence & Memorization



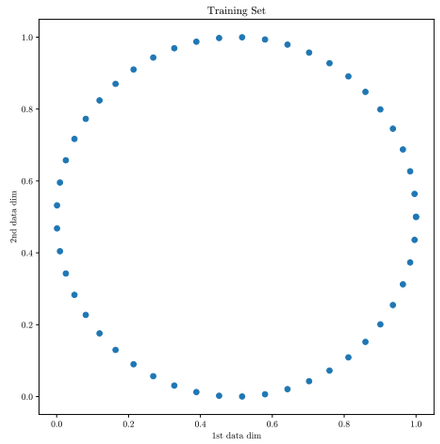
- And the other problem is you need to be cautious about the model convergence, when you have too small dataset versus too capable model, and you train it for long enough, the DM samples are no longer like the smooth density before, it’s getting closer and closer to a mixture of dirac deltas and then all it’s doing is retrieval not 'generation'
- This "converge to retrieval" has a name for it called memorization — and the worst case is your model is capable enough to just memorizing these pairs of data/ bijection, then it’s neither incorporating this similarity (recurrence) in the input space nor generating any novel samples in the output space

Recurrence & Memorization



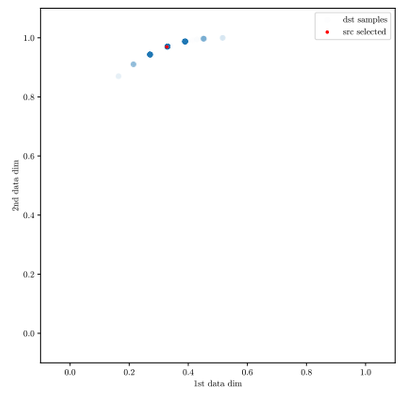
- The way we thought of to deal with this non-perfect recurrence 'or' essentially make it more generalizable in the input conditioning space — is to add gaussian noise during training

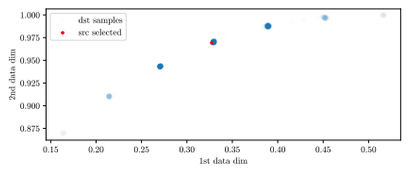
Recurrence & Memorization



- Practically it seems to work well — here is yet another synthetic data where you want to conditionally map the point on the circle to itself, so that's a bijection of data pairs.
- When you directly use vanilla training scheme, the model will memorizes the pair, i.e., condition on some point you will only get samples as the same point

Recurrence & Memorization





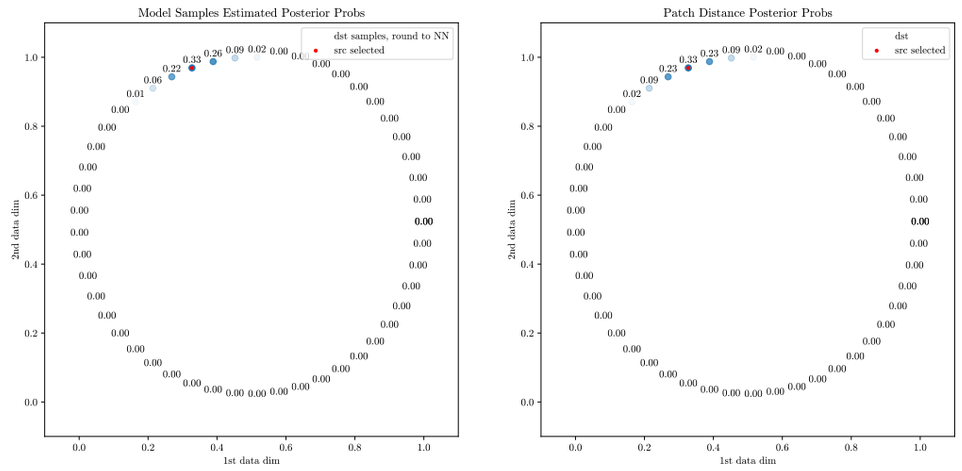
- But when you do training with gaussian noise adding to conditioning input, you no longer suffer from this bijection you can generate a distribution that looks almost like some local kernel, when given a the red dots as conditioning

Recurrence & Memorization

$$
\begin{aligned}
\Pr[\mathbf{x}_{\mathrm{dst}} = {\mathbf{x}_{\mathrm{dst}}}^{(i)} \,|\, \mathbf{x}_{\mathrm{src}} = \mathbf{x}_{\mathrm{src}}{}'] &= \frac{\Pr[\mathbf{x}_{\mathrm{src}} = \mathbf{x}_{\mathrm{src}}{}' \,|\, \mathbf{x}_{\mathrm{dst}} = {\mathbf{x}_{\mathrm{dst}}}^{(i)}] \Pr[\mathbf{x}_{\mathrm{dst}} = {\mathbf{x}_{\mathrm{dst}}}^{(i)}]}{\Pr[\mathbf{x}_{\mathrm{src}} = \mathbf{x}_{\mathrm{src}}{}']} \\
&= \frac{\Pr[\mathbf{x}_{\mathrm{src}} = \mathbf{x}_{\mathrm{src}}{}' \,|\, \mathbf{x}_{\mathrm{dst}} = {\mathbf{x}_{\mathrm{dst}}}^{(i)}] \Pr[\mathbf{x}_{\mathrm{dst}} = {\mathbf{x}_{\mathrm{dst}}}^{(i)}]}{\sum_{j=1}^{N} \Pr[\mathbf{x}_{\mathrm{src}} = \mathbf{x}_{\mathrm{src}}{}' \,|\, \mathbf{x}_{\mathrm{dst}} = {\mathbf{x}_{\mathrm{dst}}}^{(j)}] \Pr[\mathbf{x}_{\mathrm{dst}} = {\mathbf{x}_{\mathrm{dst}}}^{(j)}]} \\
&= \frac{\Pr[\mathbf{x}_{\mathrm{src}} = \mathbf{x}_{\mathrm{src}}{}' \,|\, \mathbf{x}_{\mathrm{dst}} = {\mathbf{x}_{\mathrm{dst}}}^{(i)}](1/N)}{\sum_{j=1}^{N} \Pr[\mathbf{x}_{\mathrm{src}} = \mathbf{x}_{\mathrm{src}}{}' \,|\, \mathbf{x}_{\mathrm{dst}} = {\mathbf{x}_{\mathrm{dst}}}^{(j)}](1/N)} \\
&= \frac{\Pr[\mathbf{x}_{\mathrm{src}} = \mathbf{x}_{\mathrm{src}}{}' \,|\, \mathbf{x}_{\mathrm{dst}} = {\mathbf{x}_{\mathrm{dst}}}^{(i)}]}{\sum_{j=1}^{N} \Pr[\mathbf{x}_{\mathrm{src}} = \mathbf{x}_{\mathrm{src}}{}' \,|\, \mathbf{x}_{\mathrm{dst}} = {\mathbf{x}_{\mathrm{dst}}}^{(j)}]} \\
&= \frac{\mathcal{N}\left(\mathbf{x}_{\mathrm{src}}{}'; {\mathbf{x}_{\mathrm{src}}}^{(i)}, {\sigma_{\mathrm{aug}}}^2 \mathbf{I}\right)}{\sum_{j=1}^{N} \mathcal{N}\left(\mathbf{x}_{\mathrm{src}}{}'; {\mathbf{x}_{\mathrm{src}}}^{(j)}, {\sigma_{\mathrm{aug}}}^2 \mathbf{I}\right)} \\
&= \frac{\exp\left(-\frac{1}{2{\sigma_{\mathrm{aug}}}^2} \left\| \mathbf{x}_{\mathrm{src}}{}' - {\mathbf{x}_{\mathrm{src}}}^{(i)} \right\|_2^2\right)}{\sum_{j=1}^{N} \exp\left(-\frac{1}{2{\sigma_{\mathrm{aug}}}^2} \left\| \mathbf{x}_{\mathrm{src}}{}' - {\mathbf{x}_{\mathrm{src}}}^{(j)} \right\|_2^2\right)}
\end{aligned}
$$

- However, we found this scheme (noise augmentation to conditioning) has a tight connection to the classical patch-distances method (though we use generative model)
- You can derive it using bayes rule (but I will skip it for time)

Recurrence & Memorization



- And you can also verify this equivalence with some simulation in simple 2D data.
- The probability distribution gained from the two method are almost the same
- This connection between two class of methods is more about a discovery from experiment but we’re not sure about if we can use this to "stand out" (still, theoratically, generative model can sample novel data if you don’t train it converged to memorization)

Takeaway

- We are doing (joint) patch distribution modeling using diffusion models
- Our setting of modeling joint distribution *p*(patch1, patch2, relationship) has not been explored before. There are some evidence from synthetic data that there is hope to make it work well for SR and displacement estimation.
  - More excitedly would be having a cooler killer application, what we are working on
- On the side, we found a connection between classical search-based algorithms versus parametric/deep generative method in our context — the noise augmentation setting. We're still thinking about how could we better incorporate this understanding.

- That's about it, here are some takeaway
- We are doing (joint) patch distribution modeling using diffusion models
- Our setting of modeling joint distribution p(patch1, patch2, relationship) has not been explored before. There are some evidence from synthetic data that there is hope to make it work well for SR and displacement estimation. (And, we are also thinking about other cool application that help us better sell the joint distribution setting)
- On the side, we found a connection between classical search-based algorithms versus parametric/deep generative method — the noise augmentation setting. We're still thinking about how could we better incorporate this understanding.