Diff+STN architectures for external orientation correction[⋆]

Shashwat Gupta[1[0009−0003−8037−2348]], Vidit Singh[2], and Mathieu Salzmann[3]

[1] IIT Kanpur (Indian Institute of Technology - Kanpur), India
shashwatg20@iitk.ac.in, guptashashwatme@gmail.com
[2] EPFL (Ecole Polytechnique Federale de Lausanne) vidit.vidit@epfl.ch
[3] EPFL (Ecole Polytechnique Federale de Lausanne) mathieu.salzmann@epfl.ch

Abstract. STNs are highly efficient in warping the input image for a downstream task. However, cascaded STNs are found to be able to learn more complex transformations. We attempt to leverage the multistep process of diffusion models to produce module(s) that has a similar effect to cascaded STNs.

Keywords: STN · Diffusion Models

1 Introduction

In this paper, we aim to combine Diffusion Models into the STN architecture to predict better the affine transformation applied to an image on-the-fly. Though there are many ways to combine diffusion models with STNs, we specifically focus on combinations that result in a single module that can be suited for any downstream task.

It has been shown that a series combination of STNs is quite useful for learning complex transformations. Motivated by this, we explore ways to combine STN and Diffusion Models. We also explain the process of diffusion models and STNs and suggest possible future directions and extensions of the work.

2 Diffusion Models

There are several types of generative models popular now (as shown in Figure 1), but none is without its flaws:

1. Generative Adversarial Networks (GANs): suffer from unstable training and limited diversity (mode collapse).
2. Variational Autoencoders (VAE) [8,9,23]: relies on a surrogate loss.
3. Flow-based models: need specialized architectures to construct reversible transforms.

[⋆] Supported by organization CV Lab, EPFL.

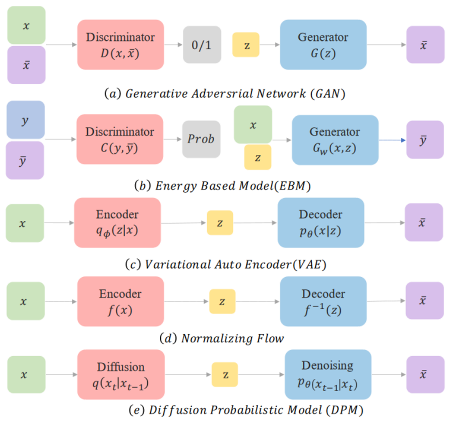

x
$\bar{x}$
Discriminator $D(x,\bar{x})$
0/1
z
Generator $G(z)$
$\bar{x}$
(a) Generative Adversrial Network (GAN)
y
$\bar{y}$
Discriminator $C(y,\bar{y})$
Prob
x
z
Generator $G_w(x,z)$
$\bar{y}$
(b) Energy Based Model(EBM)
x
Encoder $q_\phi(z|x)$
z
Decoder $p_\theta(x|z)$
$\bar{x}$
(c) Variational Auto Encoder(VAE)
x
Encoder $f(x)$
z
Decoder $f^{-1}(z)$
$\bar{x}$
(d) Normalizing Flow
x
Diffusion $q(x_t|x_{t-1})$
z
Denoising $p_\theta(x_{t-1}|x_t)$
$\bar{x}$
(e) Diffusion Probabilistic Model (DPM)

Fig. 1: Summary of Generative Models

4. Diffusion Models: inspired by non-equilibrium thermodynamics. Despite being slow at sampling, diffusion models outperform other generative models; specifically, they are free from the issues of these models.

Below, we explain the common perspectives to understand of diffusion models, specifically the ones that are needed to understand our architectures.

Markov Chain Perspective We touch upon the necessary mathematical details of the diffusion models without diving into the proofs much (More detailed treatment can be found in [1,2,3,4]). Our approach will mostly be like Denoising Diffusion Probabilistic Model (DDPM) [5,6,7] with some improvements suggested in papers published by OpenAI later on [11,12].

Diffusion Models are latent space models that involve adding noise to a sample as a Markov chain and then denoising the noisy image using a neural network. During training, noise is added (according to a variance schedule), and a model is used to denoise the image in multiple steps. During inference, denoising is applied to an isotropic noisy sample. Noising and denoising in steps, as opposed to single steps like GANs, leads to more tractable computations [10].

The forward process is defined as follows:

$$x_0 \sim q(x)$$
$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$
$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$

As $t \to \infty$, x_t approaches an isotropic Gaussian.

For the forward process, x_t can be computed in closed form from x_0 by using a reparametrization trick involving the sum of two Gaussian.

Defining two new variables:

$$\alpha_t = 1 - \beta_t$$

$$\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$$

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t} x_{t-1}, (1 - \bar{\alpha}_t)I)$$

Since β_t is small, $q(x_{t-1}|x_t)$ is also Gaussian. However, estimating this quantity would require using the entire dataset, so we learn a model p_θ to approximate the conditional probabilities.

We run the reverse diffusion process:

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$$

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

We use simple likelihood for the loss: $-\log p_\theta(x)$. Similar to VAEs, we use the variational lower bound to upper bound of the objective [8,9,**?**]. Upon simplification and additional conditioning on x_0 (for better sampling), and ignoring pure $q(x_t)$ terms (since they have no learnable parameters), we come up with the following objective:

$$\mathcal{L}_{\text{reduced}} = \sum_{t=2}^{T} D_{KL}(q(x_t|x_t, x_0)||p_\theta(x_{t-1}|x_t)) - \log(p_\theta(x_0|x_1))$$

$$p_\theta(x_0|x_1) \sim \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t); \sigma_\theta(x_t, t))$$

$$q_\theta(x_{t-1}|x_1, x_0) \sim \mathcal{N}(x_{t-1}; \tilde{\mu}_t(x_t, x_0); \tilde{\beta} I)$$

By observing that β_t is fixed (as per the schedule), as an objective, we can minimize the MSE between $\tilde{\mu}_t(x_t, x_0)$ and $\mu_\theta(x_t)$. After simplification, this reduces to the MSE between the error at time t and the predicted error for time t predicted by the model, with a scaling term that improves sample quality.

$$\mathcal{L}_{\text{simple}} = ||\epsilon_t - \epsilon_\theta(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 \bar{-} \alpha_t} \epsilon, t)||^2$$

Langevin Dynamics Perspective (Noise-conditioned score networks) This perspective enables us to understand Conditional image generation. Again we touch upon the results (more can be followed from [13,19,20,21,24]) Stochastic Gradient Langevin Dynamics [26] can generate samples from a probability density $p(x)$ using only the gradients $\nabla_x \log p(x)$ in a Markov chain of updates.

$$x_t = x_{t-1} + \frac{\delta}{2} \nabla_x \log p(x_{t-1}) + \sqrt{\delta} \epsilon_t, \text{ where } \epsilon_t \sim \mathcal{N}(0, I)$$

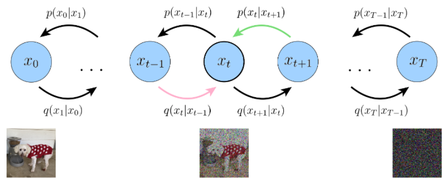


Fig. 2: Diffusion Models demystified

Here, δ represents the step size. As $T \to \infty$, $\epsilon \to 0$, and x_T converges to the true probability density $p(x)$.

Compared to standard SGD, stochastic gradient Langevin Dynamics injects Gaussian noise into the parameter updates to avoid collapsing into local minima.

Song and Ermon (2019) [13] proposed score-based generative modelling methods where samples are produced via Langevin dynamics using gradients of the data distribution estimated with Stein score-matching.

To scale with high-dimension, they add a pre-specified small noise to the data and estimate the data point with score matching. According to the manifold hypothesis, most data is expected to lie on a low-dimensional manifold, even though the data might seem to be in high dimension. Thus, the data does not cover the entire space, and estimation is unreliable in sparse regions. Adding a small perturbation in steps to cover the entire space offers more stable training.

$$\mathbf{s}_\theta(\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t} \log q(\mathbf{x}_t) = \mathbb{E}_{q(\mathbf{x}_0)} \left[\nabla_{\mathbf{x}_t} q(\mathbf{x}_t \mid \mathbf{x}_0)\right] = \mathbb{E}_{q(\mathbf{x}_0)} \left[-\frac{\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}{\sqrt{1-\bar{\alpha}_t}}\right] = -\frac{\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}{\sqrt{1-\bar{\alpha}_t}}$$

Architecture and Algorithm The original implementation of DDPMs used U-Net architecture consisted of Wide ResNet blocks, group normalisation as well as self-attention blocks. The diffusion time step t is specified by adding a sinusoidal position embedding into each residual block. Various other approaches and architectures are covered in [15,18]

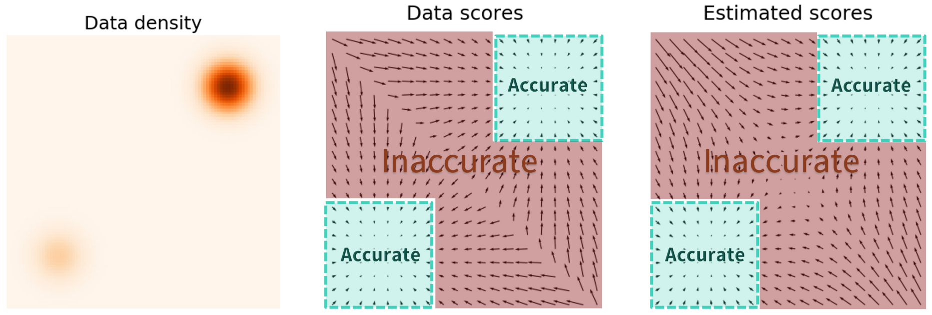
The training and Sampling algorithms are shown in Figure 4.

2.1 Conditioned Generation

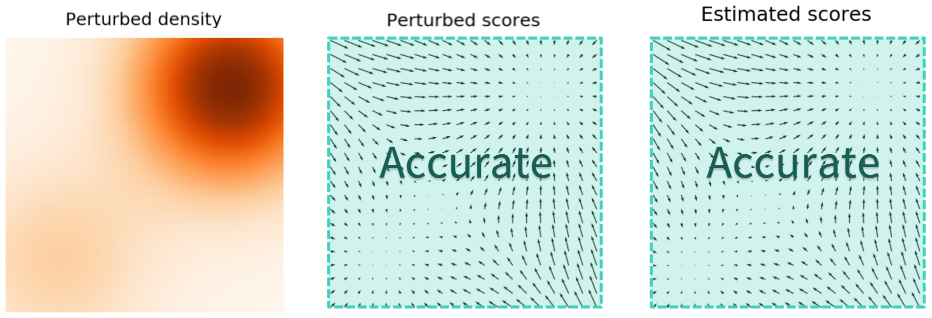
To turn a diffusion model into a conditioned model [22], we can add conditioning information (y) at each step with a guidance-scalar s as :

$$p_\theta(\mathbf{x}_{0:T}|y) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, y)$$

$$\nabla_{x_t} \log p_\theta(x_t|y) = \nabla_{x_t} \log p_\theta(x_t) + s.\nabla_{x_t} \log p_\theta(y|x_t)$$



(a) Without Noise, predictions of sparse regions are inaccurate



(b) Adding noise increases the base of predictions to sparse regions closer to the low-dimensional manifold.

Fig. 3: Role of Noise in Score-matching approach

Algorithm 1 Training

```
1: repeat
2:   $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3:   $t \sim \text{Uniform}(\{1, \ldots, T\})$
4:   $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5:   Take gradient descent step on
         $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$
6: until converged
```

Algorithm 2 Sampling

```
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: for $t = T, \ldots, 1$ do
3:   $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4:   $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: end for
6: return $\mathbf{x}_0$
```

Fig. 4: Training and Sampling algorithms for DDPM

Using $\nabla_{\mathbf{x}_t} \log q(\mathbf{x}_t) = -\frac{1}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$

$$\bar{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t, t) = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) - \sqrt{1-\bar{\alpha}_t} \nabla_{x_t} \log p_\theta(y|x_t))$$

The above score-based formulation eliminates the term using $p_\theta(y)$, which needs knowledge of all data points.

The following are the popular ways to condition the diffusion model

Classifier Guided Diffusion The score of y wrt x can be estimated using a classifier [11]. Setting $\nabla_{\mathbf{x}_t} \log q(y|\mathbf{x}_t) = \nabla_{\mathbf{x}_t} \log f_\phi(y|\mathbf{x}_t)$

$$\bar{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t, t) = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) - \sqrt{1-\bar{\alpha}_t}\, w \nabla_{\mathbf{x}_t} \log f_\phi(y|\mathbf{x}_t)$$

The resulting ablated diffusion model (ADM) and the one with additional classifier guidance (ADM-G) can achieve better results than state-of-the-art generative models (e.g., BigGAN).

Classifier-free guidance Conditioning is also possible without a classifier [17]. Let unconditional denoising diffusion model $p_\theta(\mathbf{x})$ parameterized through a score estimator $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ and the conditional model $p_\theta(\mathbf{x}|y)$ parameterized through $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y)$. These two models can be learned via a single neural network. Precisely, a conditional diffusion model $p_\theta(\mathbf{x}|y)$ is trained on paired data $(\mathbf{x}, y)$, where the conditioning information y gets discarded periodically at random such that the model knows how to generate images unconditionally as well, i.e. $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y = \phi)$.

The gradient of an implicit classifier can be represented with conditional and unconditional score estimators. Once plugged into the classifier-guided modified score, the score contains no dependency on a separate classifier.

$$\nabla_{\mathbf{x}_t} \log p(y|\mathbf{x}_t) = \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|y) - \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t)$$

$$= -\frac{1}{\sqrt{1-\bar{\alpha}_t}} \Big(\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y) - \sqrt{1-\bar{\alpha}_t}\, w \nabla_{\mathbf{x}_t} \log p(y|\mathbf{x}_t) \Big)$$

i.e.

$$\bar{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t, t, y) = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y) + w\big(\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y) - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\big) = (w+1)\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y) - w\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$$

ControlNets Zhang et al., 2023 [27] developed ControlNet, a separate module that can be added to an unconditional model for conditional image generation.

2.2 Improvements to Diffusion Model

We now discuss some popular improvements to the diffusion models:

1. Ho et al. (2020) [5] used a linear schedule from $\beta_1 = 10^{-4}$ to $\beta_t = 0.02$. Nichol and Dhariwal (2021) [11] proposed a cosine-based variance schedule (any arbitrary schedule will work as long as it offers a near-linear drop in the middle of training and subtle changes around $t = 0$ and $t = T$).
2. The DDPM paper [5] also introduced a positional time step embedding, where half of the dimensions encode sine embedding and the other half encode cosine embedding.
3. They also proposed learning the reverse process variance Σ_θ as an interpolation between β_t and $\tilde{\beta}_t$, which gives:

$$\Sigma_\theta(x_t, t) = \exp(v \log \beta_t + (1 - v) \log \tilde{\beta}_t)$$

   Song et al., 2021 [28] proposed using deterministic sampling (Denoising Diffusion implicitly model - DDIM 2020), which has the same marginal noise distribution but deterministically maps noise back to the original data samples. Compared to DDPM, DDIM has higher sample quality for small steps, consistency of high-level features on conditioning and thus, the semantically meaningful representation of a latent variable.
4. Nicol and Dhariwal (2021) [11] also proposed speeding up diffusion process by strided sampling.
5. Latent Diffusion [27] runs the diffusion process in latent space instead of pixel space, thus lower training cost and faster inference. The encoder downsamples to latent space, and the decoder is used to recover back the generated image.
6. Cold Diffusion [14], generalises the notion of noise by applying various transformations to the image. and uses a modified sampling algorithm to make the degradation function independent of the restoration operator up to first-order terms.

3 Spatial Transformer Networks

Much of this section is inspired by the original paper on Spatial Transformer Networks [30,31]

STNs (Spatial Transformer Networks) are learnable modules to actively manipulate spatial information and make the model more robust to warping. before STNs, this could only be achieved by a long hierarchy of Max-Pooling layers. Unlike pooling layers, where the receptive fields are fixed and local, the spatial transformer module is a dynamic mechanism that can actively spatially transform an image (or a feature map) by producing an appropriate transformation for each input sample. The transformation is performed on the entire feature map (non-locally) and can include scaling, cropping, rotations, and non-rigid deformations. This allows networks to select regions that include spatial transformers

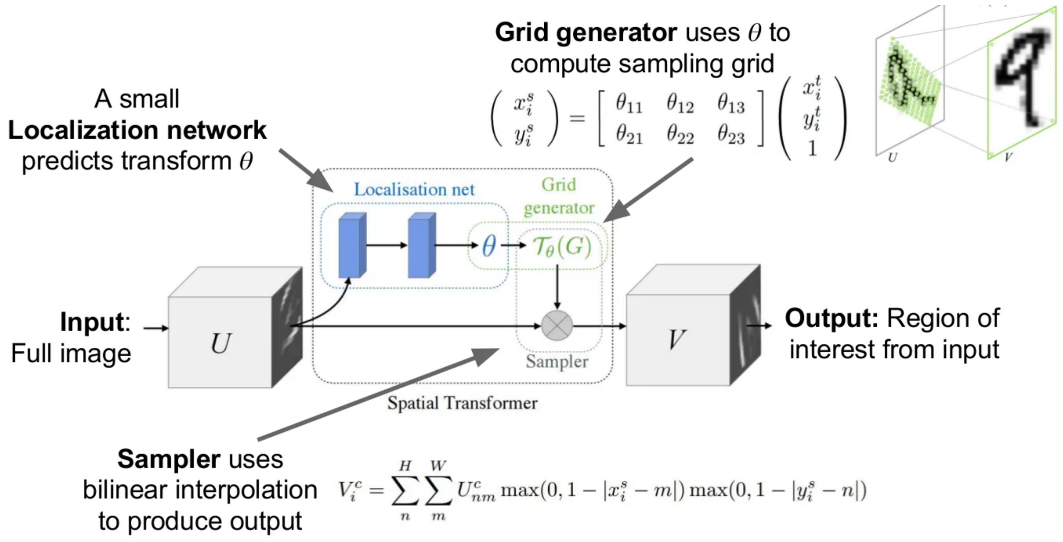

Grid generator uses θ to compute sampling grid
$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$
A small **Localization network** predicts transform θ
Localisation net
Grid generator
θ
$\mathcal{T}_\theta(G)$
Input: Full image
U
V
Output: Region of interest from input
Sampler
Spatial Transformer
Sampler uses bilinear interpolation to produce output
$V_i^c = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|)$

Fig. 5: The figure shows the working of the STN module

to not only select regions of an image that are most relevant (attention) but also to transform those regions to a canonical, expected pose to simplify recognition in the following layers. They are a generalisation of differential attention modules on spatial transformation. They can be trained with standard back-propagation, allowing for end-to-end training of the models they are injected in. They are useful for various tasks, including image classification, co-localisation, and spatial attention

The STN Consists of the following 3 parts:

1. Localisation Net
2. Grid Generator
3. Sampler

Localisation Network It takes the input feature map $U \in \mathbb{R}^{H*W*C}$, and outputs the parameters of transformation ($\theta = f_{\text{loc}}(U)$). It can take any form but should include a final regressor layer to produce the transformation parameters θ

Parametrised Grid Sampling The output pixels are computed by applying a sampling kernel centred at each location of the input feature map. The only constraint is that the transformation should be different wrt the parameters to allow for back-propagation. A good heuristic is to predict the transformation parametrised in a low dimensional way so that the complexity of the task assigned to the localisation network is reduced, and it can also learn about the target grid representation. e.g. if $\tau_\theta = M_\theta B$, where B is the target representation. Thus, it is also possible to learn θ and B.

In our case, we analyze 2D transformations, which the following equation can overall summarise:

$$\begin{pmatrix} x_s^i \\ y_s^i \end{pmatrix} = \tau_\theta(G_i) = A_\theta \begin{pmatrix} x_t^i \\ y_t^i \\ 1 \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_t^i \\ y_t^i \\ 1 \end{pmatrix}$$

Here, (x_t^i, y_t^i) are target coordinates of the regular grid in the output feature map, and (x_s^i, y_s^i) are the source coordinates. We use height and width normalised coordinates.

Differentiable Image Sampling Differentiable Image Sampling

To perform a spatial transformation of the input feature map, a sampler must take the set of sampling points $T_\theta(G)$, along with the input feature map U, and produce the sampled output feature map V. Each (x_i^s, y_i^s) coordinate in $\tau_\theta(G)$ defines the spatial location in the input where a sampling kernel is applied to get the value at a particular pixel in the output V. This can be written as

$$V_i^c = \sum_{n=1}^{H} \sum_{m=1}^{W} U_{nm}^c k(x_{si} - m; \Phi_x) k(y_i^s - n; \Phi_y) \quad \forall i \in [1, H'W'] \quad \forall c \in [1, C]$$

where Φ_x and Φ_y are the parameters of a generic sampling kernel $k()$ which defines the image interpolation (e.g. bilinear), U_{nm}^c is the value at location (n, m) in channel c of the input, and V_i^c is the output value for pixel i at location (x_i^t, y_i^t) in channel c. Note that the sampling is done identically for each channel of the input, so every channel is transformed identically (this preserves spatial consistency between channels).

In theory, any sampling kernel can be used, as long as (sub-)gradients can be defined with respect to x_i^s and y_i^s. For example, using the integer sampling kernel reduces the above equation to

$$V_i^c = \sum_{n=1}^{H} \sum_{m=1}^{W} U_{nm}^c \delta([x_i^s + 0.5] - m) \delta([y_i^s + 0.5] - n)$$

where $[x + 0.5]$ rounds x to the nearest integer and $\delta()$ is the Kronecker delta function. This sampling kernel equates to just copying the value at the nearest pixel to (x_i^s, y_i^s) to the output location (x_i^t, y_i^t). Alternatively, a bilinear sampling kernel can be used, giving

$$V_i^c = \sum_{n=1}^{H} \sum_{m=1}^{W} U_{nm}^c \max(0, 1 - |x_{si} - m|) \max(0, 1 - |y_{si} - n|)$$

To allow backpropagation of the loss through this sampling mechanism, we can define the gradients with respect to U and G. For bilinear sampling above equation, the partial derivatives are

$$\frac{\partial V_i^c}{\partial U_{nm}^c} = \sum_{n=1}^{H} \sum_{m=1}^{W} \max(0, 1 - |xs_i - m|) \max(0, 1 - |y_i^s - n|)$$

$$\frac{\partial V_i^c}{\partial x_i^s} = \sum_{n=1}^{H} \sum_{m=1}^{W} U_{nm}^c \max(0, 1 - |y_i^s - n|) \begin{cases} 0 & \text{if } |m - x_i^s| \geq 1 \\ 1 & \text{if } m \geq x_{si}^s \\ -1 & \text{if } m < x_i^s \end{cases}$$

and above equation for $\frac{\partial V_{ci}}{\partial y_{si}}$.

This gives us a (sub-)differentiable sampling mechanism, allowing loss gradients to flow back not only to the input feature map but also to the sampling grid coordinates and, therefore, back to the transformation parameters θ and localization network since $\frac{\partial x_i^s}{\partial \theta}$ and $\frac{\partial y_i^s}{\partial \theta}$ can be easily derived. Due to discontinuities in the sampling functions, sub-gradients must be used. This sampling mechanism can be implemented very efficiently on GPU by ignoring the sum over all input locations and instead just looking at the kernel support region for each output pixel.

For better warping, the STNs can be cascaded by passing the output of one STN to the next (as in [30]) and with additional input to condition (as in [29])

Overall analysis of STNs The overall pros of STNs are :

1. STNs are very fast, and the application does not require making many modifications to the downstream model
2. They can also be used to downsample or oversample a feature map (downsampling with fixed, small support might lead to an aliasing effect)
3. Multiple STNs can be used. The combination can be in Series (for more complex feature learning, with the input of one STN going into another, with or without an unwarped conditional input.
4. Parallel combinations are effective when there are more than one parts to focus on in images (It was shown that of 2 STNs used on the CUB-200-2011 bird classification dataset, one became head-detector and the other became body-detector)

However, STNs are notoriously known to suffer from the following 2 defects :

1. Boundary effect arises as the image is propagated and not the geometric information (e.g. if an image is rotated, STNs can fix the rotation, but they do not fix the degraded boundary effects like cut corners etc.)
2. Single STN application is insufficient to learn complex transformations

4 Model 1 : Diff-in-STN Architecture (predicting $\boldsymbol{\theta}$ from diffusion model based on localisation network

In this experiment, we use a conditional diffusion model to learn the parameters of the matrix θ by just utilising the transformed image.

4.1 Model architecture and Algorithm

The model architecture is defined in the following image. It essentially takes the transformed image and passes it to the image encoder. The encoder generates an embedding of the image, which is used as conditioning to the DDPM Module. The diffusion model takes encoded θ as input and learns it by noising and denoising it based on the image conditioning. This generated θ is used for the further processes of the model, like grid generation and sampling and, finally, downstream tasks (her image classification).

For the theta-encoder, we used 2 layers fully connected network, which increases the dimensions of theta from 6 to 20. We found that using Latent Space with higher dimensions allows us to learn and better generate θ. The Diffusion layer comprises 3 fully connected layers with dimensions remaining 20 for both input and output. The regressor is the dimension decoder and comprises 2 fully connected layers that reduce dimension from 20 to 6. The result from the regressor is passed to the grid generator.

For the training, we replaced algorithm 1 (in original DDPM paper) with algorithm 2 where noise steps is chosen from a uniform distribution for the batch. This is because we are developing a module and algorithm 1 is suited for seperate training of diffusion model than the downstream task. We proceed with noise predicted by neural network to denoise and predict theta, which is used to generate the image. The final objective function is the negative log likelihood loss for the classification downstream.

4.2 Data description and Augmentation

We augment the MNIST dataset from torch vision based on the affine transformation. We use the standard 2D Affine Matrix defined above. We use rotation, translation and individual dimension scaling (shear). We provide the sample with a range and sample from that range to generate the parameters for the transformation. The Affine matrix A_θ can be represented as follows:

$$\begin{bmatrix} k_x.\cos(\phi) & -k_y.\sin(\phi) & t_x \\ k_x.\sin(\phi) & -k_y.\cos(\phi) & t_y \end{bmatrix}$$

; where ϕ represents rotation (degrees), t_x and t_y represent translation of coordinates and k_x and k_y represent the rotational scaling. We generate train and test loaders to return the transformed image, the correct classifier label y and the transformation matrix A_θ.

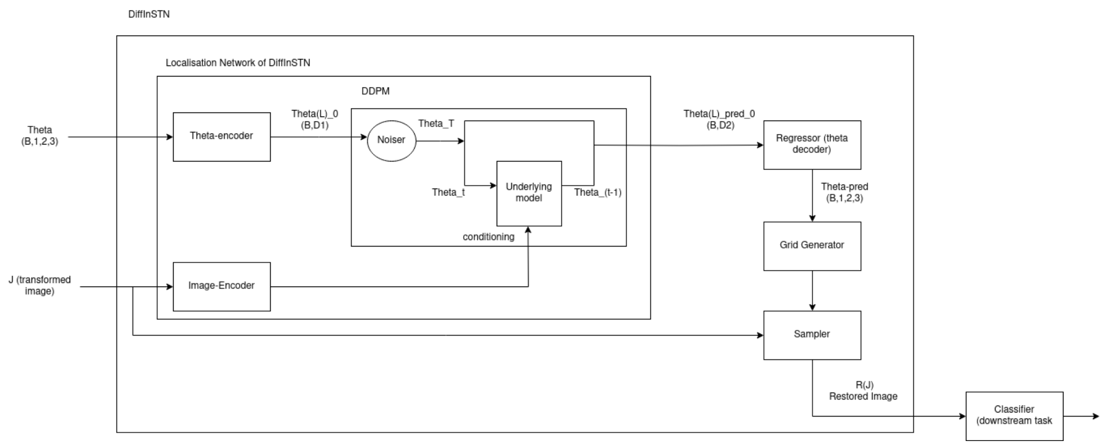


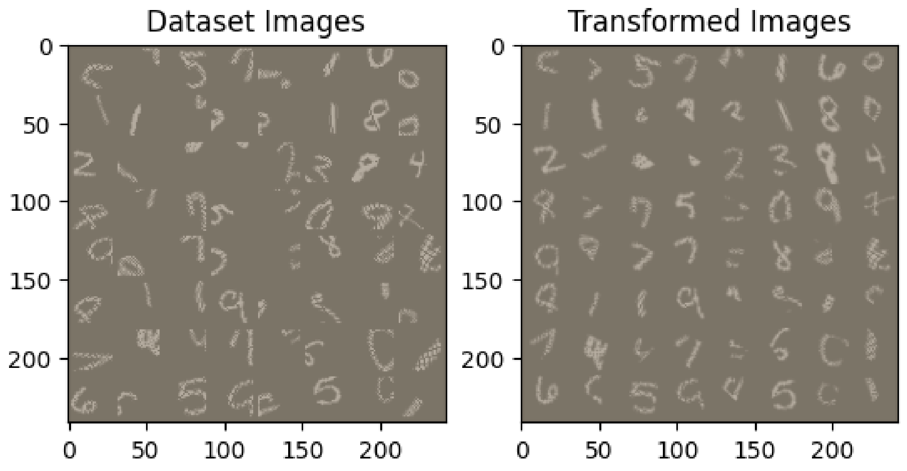
Fig. 6: The figure describes the model architecture during training. The learnable modules are represented in rectangles. During the testing, we could replace the output of Theta_encoder with a theta-shaped matrix sampled from an isotropic Gaussian

4.3 Tuning and Results

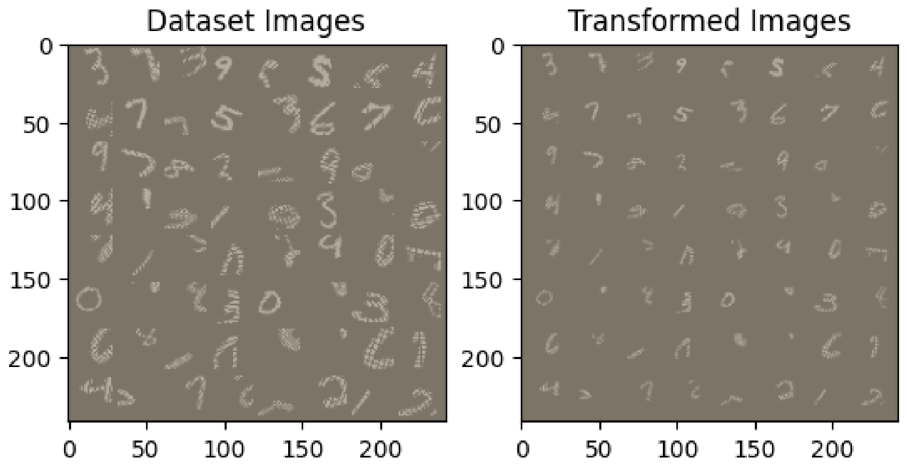
Out of the various things we tried, we present some of them which led to improvement of the model's performance. The base performance of the model was close to 20% (random is 10%). Since the search space was huge for trying out various things, we followed a heuristic-based approach, which means that we tried something if we found some mention of it in the literature that led to improvement in a similar setting. As a baseline, STNs are quite fast and have an accuracy of 84% using SGD and learning rate = 0.01. Since SGD has a stochastic component, we report an average of 3 runs.

We also note that the Differential Attention of STNS enables not to shrink the final images, whereas the final transformation of Diff-in-STN is shrunk; in general, the images corrected have a small transformation applied to them depending on the mean of transformation extremes at data-warping/augmentation stages. Also, we find that the Diff-in-STN essentially learns mostly scaling and not many other transformations, this might be because the classifier downstream might be good enough for classification, and diff-in-stn is slower to learn than the classifier. We might need to deploy a highly selective classifier downstream to train the diff-in-stn properly. Since we wished to develop a self-contained module that could be added on top of other modules, we did not explore this direction much.

ImageEncoder We considered 2 choices for ImageEncoder, on of fully connected layers and the other of CNNs. The fully connected Encoder (image-classifier type) resulted in the model being random, no matter whatever other

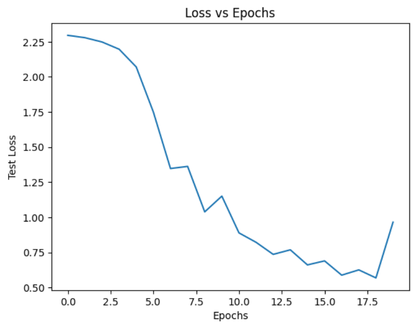


(a) Visualisation of STN output wrt input dataset

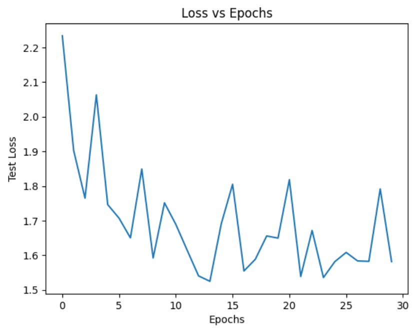


(b) Visualisation of Diff-in-STN output wrt input dataset

Fig. 7: Comparison of Output Map of STN (Figure 6) and Diff-in-STN (Figure 7)

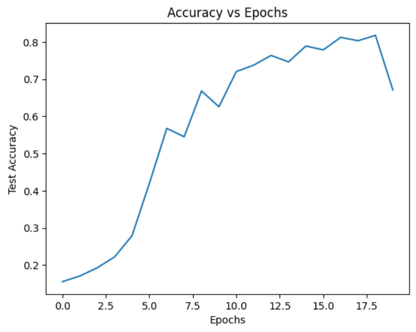


(a) Visualisation of STN loss

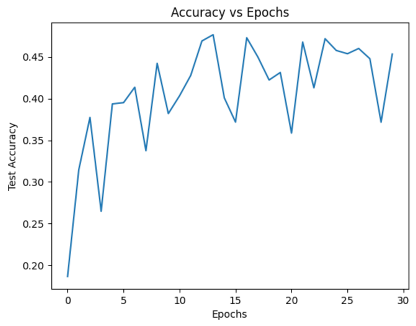


(b) Visualisation of Diff-in-STN loss

Fig. 8: Comparison of Test loss in STN vs Diff-in-STN



(a) Visualisation of STN accuracy



(b) Visualisation of Diff-in-STN accuracy

Fig. 9: Comparison of test Accuracy in STN vs Diff-in-STN

tuning we do. This makes sense, as the image is already classified at the ImageEncoder step, and the downstream classifier can't do much about it. The other choice was using a CNN-based down-sampler (similar to the original localisation block used in STN). This leads to improvement in accuracy (up to 23%). With this type of ImageEncoder, the DDPM model essentially acts to improve upon the estimated Transformation predicted by the localisation network in a higher dimensional space, followed by a regressor. We considered only 2 types of Encoders. However, any other types of encoders that generate identity+transformation representations of images could be used.

Optimizer We found using SGD (Stochastic Gradient Descent) as an Optimizer outperformed other optimizers like Adam, RMS etc. The SGD was the optimiser used in the original STN paper. It leads to noisy and slow training, but it leads to better classifier accuracy (close to 40%)

Hyperparameters tuning Adding weight decay and setting the learning rate to 0.1 helped better generalise the model. This reduced the loss to 1.6 from 1.8, and accuracy jumped from 30% to 45%. Paradoxically, lr=0.1 led to overshooting for STNs and thus poor training with very little accuracy (25% initially, after which STNs performed just like a random model with 10% accuracy)

Variance Schedule and Moving Averages Motivated by Improved DDPM, we tried to incorporate a Cosine schedule for slowly noising the images and used expected moving averages for smoother training free from outliers. This, with SGD(lr=0.1 and weight-decay=0.001), led to 50% accuracy. This is the maximum accuracy that we found from this architecture.

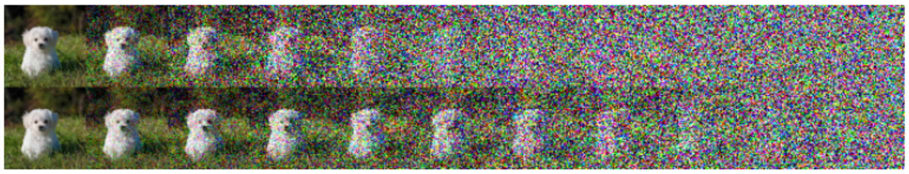


Fig. 10: Linear (top) vs Cosine variance schedule (bottom), Credits – Nicol and Dhariwal, 2021

Using a feature extraction technique We tried to use Canny edge detector from cv2 (OpenCV) but found that using this really degrades the result. This lead to a higher loss for Diff-in-STNs.

Latent Space running DDPM on higher dimension than θ enabled the model to learn complex parameters at the cost of time. Thus, the model was better at estimating the parameter but took longer to train and infer.

Passing time embedding Passing time-embedding to each layer of the DDPM layer leads to slightly faster convergence (however the convergence is to the barrier of 50%)

5 Model 2 : STN-in-Diff Architecture (predicting untransformed image from diffusion model based on localisation network

In this model architecture, we predict the most suitably oriented image for classification. The UNet is used as the model for diffusion model and we use localisation based conditioning for the diffusion model.

Model Architecture and Algorithm The model comprises of DDPM with unet conditioned on the output of image encoder. The image encoder is a double downsample block with maxpooling layers and activation function. This is just like the localisation network in the original STN paper.

Just like Architecture 1, we use Modified Algorithm 2 for Sampling. Since there are relatively few learnable parameters, there is not much to tune in this architecture.

We did not optimise much on the inferences nor try to encode the images to a different space since there are algorithms that can do this. For fast inference and training, we could use eg Exponential Integrator [16], and to reduce the dimensionality of model (and the parameters), we could explore running Latent Diffusion [27], or simply use fixed Encoders (or other learnable modules).

5.1 Results

We found that the model was highly inefficient and could not learn well in 30 epochs. This is the known characteristic of diffusion models. This observation renders this model useless without other optimisation in algorithms (training and sampling) and dimensions (latent space)

6 Conclusion

We found that the Diff-in-STN is actually a potential model, with the optimisations. The classifier accuracy was close to 50% for our runs, and we hope that we could reach it to 80% or more, than the STNs.

We found that the second architecture was highly inefficient and, for the given 30 epochs, was both slow and as bad as random. This suggests that our original

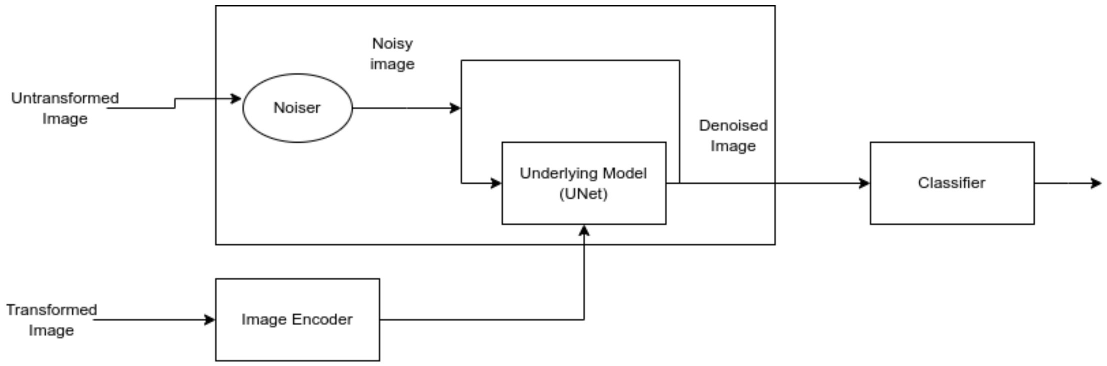


Fig. 11: The figure describes the model architecture for STN-in-Diff during training. The learnable modules are represented in rectangles. During the testing, we directly sample from an isotropic Gaussian instead of original untransformed image

model was good enough as compared to a plain diffusion model, guided by a double downsampler. However, there is now free-lunch. The first model (Diff-in-STN) needed an additional parameter θ, i.e. it needed to know the transformation beforehand, which is usually hard to get in real settings. The only possible benefits of STN-in-Diff model are :

1. It does not need the transformation knowledge, just the original images during the training.
2. With just a linear approximation of some higher order transformation, we can directly predict a better transformed image without actually estimating the transformation parameter.

However, the Diff-in-STN model is quite light and fast with a reasonable performance.

Applications We now enumerate the following applications of our research:

1. We can predict the transformations applied to an image. This is till now done using approximate algorithms,
2. The problem can be modelled to finding an inverse of a matrix. This usually needs $O(n^3)$ Complexity. If we could make the models efficient, we can estimate matrix inverses with just small knowledge about the inverse.
3. We could conclude from the transformations which orientations/transformations are suited to classifiers. Once this generalisation is made, we could apply these to increase the efficiency of classifiers (or other downstream modules).

7 Future Directions

During the research, we found several interesting directions:

1. The (STN-in-Diff) is quite simple to implement. However, it is not sure though if the only transformation to the output image is an affine transformation, and we whether algorithms like RANSAC, DLT could predict the first-order linear approximation to the transformation accurately. However, such a model is useful to get insights to the kind of transformation on dataset, which is easier for classifer to learn. We could use Latent Diffusion [27] to develop this.
2. We can also model the error to θ as distributions other than Gaussian. In this case, we might now be able to use the same algorithm for sampling, BUT we could use the modified algorithm proposed in cold-diffusion paper.
3. We could explore the use of Affine Invariants for Image Encoders(models like SymmNets an Scatter Nets [36]). Lenc et al., 2015 characterised a linear method to characterise equivariance, equivalance and invariance of affine transformations [32] .
4. Though using classifier type models for Image Encoders seemed to be a bad idea, we do not negate the possibility of utilising the top-k logits for classification. Seeing at a superficial level, this information greatly reduces the search-space of the model and hence however bad the diffusion model is, we can get accuracy of classification to be more than 10% (random). A variation could be dynamically vary the result of ImageEncoder (and hence the top-k) values for the output.
5. Utilising Feature Extraction methods (e.g. retaining low frequency features) might help the model learn betters given the same number of parameters, owing to the fact that the low-frequency features are more correlated to classes-identity
6. Utilising ensemble of diffusion models for better conditioning and image generation.
7. Above all, training the network for different datasets like CIFAR, and with higher order transformations (possiblity non-linear) transformations could be more conclusive about the working of architecture.
8. We could sample in fewer steps using aporoaches like Exponential Integrator [16]

8 Our Code

Our code can be found at `https://drive.google.com/drive/folders/1Y_Rw5K0JW-2oveibcuX1PSYDdZWqXPDy`. Most of the elementary modules and blocks are taken from : [33,34,35].

It contains the following files (preferable to run in this very order):

1. STN on Affine Transformations : `https://colab.research.google.com/drive/1DvMIaID1iGxcBl9qFB5VK_BE35-3KO8R#scrollTo=rZCCgO288q2t`
2. Diff-in-STN : `https://colab.research.google.com/drive/17ir9Eq4U6o3GHwMzNlvR83xO7eN4amtZ#scrollTo=9JcVQvvPLPih&uniqifier=1`
3. STN-in-Diff : `https://colab.research.google.com/drive/1_XJ_Dd0_dXBGGd85cYvVRSshQmwYmU34#scrollTo=LdbWDjvVn0ji`

There is a folder called Unfinished which includes some of the code that we tried for other things such as Future Directions (Specifically - Cold Diffusion to try out modelling transformations with noise other than Gaussian, and Latent Diffusion, to work on the reducing sampling time and model parameters)

Then the Folder Experiments contains the code that I used for learning (tutorials and snippets).

References

1. Blog: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/
2. Blog: https://theaisummer.com/diffusion-models/
3. Blog: https://towardsdatascience.com/diffusion-models-made-easy-8414298ce4da (a simplistic explanation)
4. Video: https://www.youtube.com/watch?v=HoKDTa5jHvg&t=1284s
5. Paper: DDPM: https://arxiv.org/pdf/2006.11239.pdf (Ho et al., 2020)
6. Video Explanation: https://www.youtube.com/watch?v=W-O7AZNzbzQ
7. Annotated Code: https://huggingface.co/blog/annotated-diffusion
8. Blog - Variational AutoEncoders: https://lilianweng.github.io/posts/2018-08-12-vae/
9. Blog - Latent Variable Models: https://theaisummer.com/latent-variable-models/
10. Paper: Deep Unsupervised Learning using Nonequilibrium Thermodynamics, Dickstein et al., 2015: https://arxiv.org/pdf/1503.03585.pdf
11. Paper: Improved Denoising Diffusion Probabilistic Models, Nicol and Dhariwal, 2021: https://arxiv.org/pdf/2102.09672.pdf
12. Paper: Diffusion Models beat GANs on Image Synthesis: https://arxiv.org/pdf/2105.05233.pdf
13. Paper: Generative Modelling by estimating Gradients of Data Distribution: Noise-conditioned score network, Yang and Ermon, 2019: https://arxiv.org/abs/1907.05600
14. Paper: Cold Diffusion: https://arxiv.org/pdf/2208.09392.pdf
15. Paper: Understanding Diffusion Models: A Unified Perspective, Calvin Luo, 2022: https://arxiv.org/pdf/2208.11970.pdf
16. Paper: Fast Sampling of Diffusion Models with Exponential Integrator, Zhang et al., 2020: https://arxiv.org/abs/2204.13902
17. Paper: Classifier-Free Diffusion Guidance (Ho et al., 2021): https://openreview.net/pdf?id=qw8AKxfYbI
18. Paper: Diffusion Models: A Comprehensive study of Methods and Applications, Yang et al., 2022: https://arxiv.org/pdf/2209.00796.pdf
19. Diffusion and Score-based generative models: https://www.youtube.com/watch?v=wMmqCMwuM2Q
20. Blog: https://yang-song.net/blog/2021/score/
21. Blog : Autoregressive models, normalizing flow, energy-based models, VAEs. Score-papers: https://scorebasedgenerativemodeling.github.io/
22. Blog: Guiding Diffusion Process: https://sander.ai/2022/05/26/guidance.html
23. Blog: Diffusion as autoencoders: https://sander.ai/2022/01/31/diffusion.html
24. Video : Langevin Dynamics end to end: https://www.youtube.com/watch?v=3-KzIjoFJy4&t=2379s

25. Paper : Adding Conditional Control to Text-to-Image Diffusion Models, Zhang et al, 2023 https://arxiv.org/abs/2302.05543
26. Paper : Bayesian Learning vis Stochastic Gradient Langevin Dynamics, Welling and Teh, 2011 https://www.stats.ox.ac.uk/~teh/research/compstats/WelTeh2011a.pdf
27. Paper : High-Resolution Image Synthesis with Latent Diffusion Models Rombach et al., 2022 https://arxiv.org/abs/2112.10752
28. Paper : Denoising Diffusion Implicit Models, Song et al.,2021 https://arxiv.org/abs/2010.02502
29. Paper: IC-STN: https://arxiv.org/pdf/1612.03897.pdf
30. STN: https://paperswithcode.com/method/stn
31. Video: https://www.youtube.com/watch?v=6NOQC_fl1hQ&t=162s (with slides, CV reading group resources)
32. Paper: Lenc and A. Vedaldi. Understanding image representations by measuring their equivariance and equivalence. CVPR, 2015 (defines affine invariance, equivariance, and equivalence criterion)
33. DDPM Model: https://github.com/dome272/Diffusion-Models-pytorch (Video: https://www.youtube.com/watch?v=TBCRlnwJtZU)
34. Annotated Implementation: https://huggingface.co/blog/annotated-diffusion
35. STN - PyTorch Implementation: https://pytorch.org/tutorials/intermediate/spatial_transformer_tutorial.html
36. Scatter Nets: https://paperswithcode.com/paper/invariant-scattering-convolution-networks#code