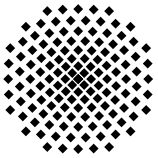

University of Stuttgart
Institute for Signal Processing and System Theory
Professor Dr.-Ing. B. Yang

Masterarbeit Dxxxx TBD

6 DoF Pose Esitmation with Diffusion Models from single RGB-D Image

Arbeitstitel, to be defined (TBD)

Author: Student's name TBD

Date of work begin: Date of work begin TBD
Date of submission: Date of submission TBD

Supervisor: Supervisor's name TBD

Keywords: Keyword1, Keyword2 TBD

Abstract TBD

Contents

1. Introduction

1.1. Motivation

1.2. Main Objective

1.3. Structure of the Thesis

As shown in [1], we present an equation

$$H(\omega) = \int h(t)\, \mathrm{e}^{\mathrm{j}\,\omega t}\, \delta t \in \mathbb{N} \tag{1.1}$$

Then we include a graphic in figure 1.1 and information about captions in table 1.1.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Figure 1.1.: A beautiful mind

Table 1.1.: Where to put the caption

	above	below
for figures	no	yes
for tables	yes	no

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

2. Background

2.1. 6 DoF Pose Estimation

2.1.1. Definition

Six degree-of-freedom(DoF) pose refers to the six degrees of freedom of movement of a rigid body in three-dimensional space. Especially, it represents the freedom of a rigid body to move in three perpendicular directions, called translations, and to rotate about three perpendicular axes, called rotations. This concept is widely applied in the industial and automotive field to measure and analyze the spacial properties of objects.

In domain of computer vision and robotics, 6 DoF pose estimation is a fundamental task that aims to estimate the 3D translation $\mathbf{t} = (t_x, t_y, t_z)$ and rotation $\mathbf{R} = (\Phi_x, \Phi_y, \Phi_z)$ of an object related to a canonical coordinate system using the sensor input, such as RGB or RGB-D data[2].The object M is typically a known 3D CAD model, consisting of a set of vertices $V = \{v_1, ..., v_N\}$, with $v_i \in \mathbb{R}^3$ and $V \in \mathbb{R}^{3\times N}$ and triangles $E = \{e_1, ..., e_M\}$, with $e_i \in \mathbb{R}^3$ and $E \in \mathbb{R}^{3\times M}$ connecting the vertices. Furthermore, if the query image is a multi-object scenario with N objects $O = \{M_1, ..., M_N\}$, we need to detect and estimate the pose of each object M_i in the image[3]. Following figure 2.1 shows a general structure of the learning-based 6 DoF pose estimation.

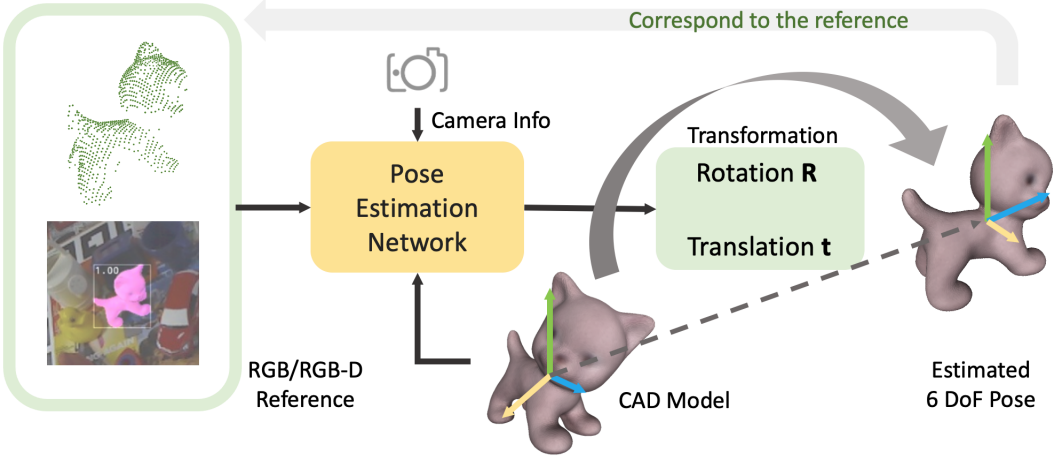

Figure 2.1.: Overview of the 6 DoF pose estimation

2.1.2. Representing 6 DoF Pose

6 DoF pose can be treated seperately as 3D translation and 3D rotation. The 3D translation is simply represented by 3 scalars along the X, Y, and Z axis of the canonical coordinate system. We can use either the deep learning methods to estimate the depth and the corresponding 2D projection from RGB images or even get the depth information fused from RGB-D data[4]. After that, the object can be shifted back to the camera coordinate system by adding translation vector to the object vertices V

$$V^{'} = V + \mathbf{t} \tag{2.1}$$

Similarly, the 3D rotation can be represented by 3 rotation matrics around the X, Y and Z axis. And rotating the object vertices V by the rotation matrix $\mathbf{R}_i$ with $i \in \{X, Y, Z\}$ can be achieved by multiplying them. Rotation around X axis is defined as

$$V^{'} = \mathbf{R}_X(\Phi_x)V = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\Phi_x) & -sin(\Phi_x) \\ 0 & sin(\Phi_x) & cos(\Phi_x) \end{bmatrix} V \tag{2.2}$$

Rotation matrix $\mathbf{R}_Y$ and $\mathbf{R}_Z$ can be defined repectively with

$$\mathbf{R}_Y(\Phi_y) = \begin{bmatrix} cos(\Phi_y) & 0 & sin(\Phi_y) \\ 0 & 1 & 0 \\ -sin(\Phi_y) & 0 & cos(\Phi_y) \end{bmatrix} \tag{2.3}$$

$$\mathbf{R}_Z(\Phi_z) = \begin{bmatrix} cos(\Phi_z) & -sin(\Phi_z) & 0 \\ sin(\Phi_z) & cos(\Phi_z) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

The rotation matrix $\mathbf{R}$ can be obtained by multiplying the three rotation matrices $\mathbf{R}_X$, $\mathbf{R}_Y$ and $\mathbf{R}_Z$ together, but changing the order of the multiplication will result in different rotation matrix. The common order is defined a $Z - Y - X$ order, which means the rotation around X axis is performed first, then Y axis and finally Z axis. All possible rotations in 3D Euclidean space establish a natual manifold known as special orthognal group $\mathbb{SO}(3)$[5].

Togather with the translation vector $\mathbf{t}$, the 6 DoF pose can be represented by a 4x4 transformation matrix $\mathbf{T}$ as

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \in \mathbb{SE}(3) \tag{2.5}$$

The partitioned transformation matrix with 3x3 rotation matrix $\mathbf{R}$ and a column vector $\mathbf{t}$ that represents the translation is also called homogeneous representation of a transformation. All possible transformation matrices of this form generate the special Euclidean group $\mathbb{SE}(3)$

$$\mathbb{SE}(3) = \{\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} | \mathbf{R} \in \mathbb{SO}(3), \mathbf{t} \in \mathbb{R}^3\} \tag{2.6}$$

Normally, we use the shift in 3 orthognal directions in cartesian coordinate system to represent the translation. However there are some different ways to represent the rotation.

One simple method to represent the rotation is to use the Euler angles ϕ, θ and ψ which are also marked as roll angle (around X axis), pitch angle (around Y axis) and yaw angle (around Z axis) respectively. The main drawback of this representation is the gimbal lock problem, which means the rotation around two axes will cause the rotation around the third axis to be the same as the rotation around the first axis.

An alternative representation of 6 DoF pose is a 4-dimensional vector that consists of translation and rotation quaternion which has a compacter form

$$\mathbf{r} = (q_w, q_x, q_y, q_z)^T \tag{2.7}$$

Where the quaternion q is defined as

$$q = q_w + q_x i + q_y j + q_z k \quad \text{with} \quad i^2 = j^2 = k^2 = ijk = -1 \tag{2.8}$$

Normally, regressing the rotation matrix directly is not a common choice since the same rotation can be achieved via different combinations of Euler angles. And the unit quaterion form is in many case prefered because it can ensure the uniqueness by restricting the quaterion on the upper hemisphere of $q_w = 0$ plane and can also guarantee a gimbal-lock free rotation in $\mathbb{SO}(3)$[6].

Another representation that can be considered is called modified Rodrigues parameters (MRPs) which is a 3-dimensional vector $\mathbf{r} = (r_1, r_2, r_3)^T$. They are triplets in $\mathbb{R}^3$,bijectively and rationally mapped to quaternions through stereographic projection[7]. The MRP vector $\mathbf{r}$ is defined as

$$\mathbf{r} = \frac{\mathbf{q}}{1 + q_w} = \frac{1}{1 + q_w}(q_x, q_y, q_z)^T \tag{2.9}$$

where $\mathbf{q}$ is the quaternion representation of the rotation. The MRP vector $\mathbf{r}$ is also a unit vector, the advantage of using MRP is that a random assignment of the vecter within the unit sphere will always result in a valid rotation. That property makes this representation more robust in the forward and reverse process of the diffusion pipeline.

Zhou et al.[8] proposed a novel representation of rotation called 6D continuous rotation representation. The mapping from the rotation matrix to the 6D representation with generally n dimensional rotation is defined as:

$$g_{GS}\left(\begin{bmatrix} | & & | \\ a_1 & \cdots & a_n \\ | & & | \end{bmatrix}\right) = \begin{bmatrix} | & & | \\ a_1 & \cdots & a_{n-1} \\ | & & | \end{bmatrix} \tag{2.10}$$

The mapping from $\mathbb{SO}(3)$ to the 6D representation can be simplified as:

$$g_{GS}\left(\begin{bmatrix} | & | & | \\ a_1 & a_2 & a_3 \\ | & | & | \end{bmatrix}\right) = \begin{bmatrix} | & | \\ a_1 & a_2 \\ | & | \end{bmatrix} \tag{2.11}$$

The reverse mapping follows the Gram-Schmidt-like process:

$$f_{GS}\left(\begin{bmatrix} | & | \\ a_1 & a_2 \\ | & | \end{bmatrix}\right) = \begin{bmatrix} | & | & | \\ b_1 & b_2 & b_3 \\ | & | & | \end{bmatrix} \tag{2.12}$$

$$b_i = \begin{bmatrix} \begin{cases} N(a_1) & \text{if } i = 1 \\ N(a_2 - (b_1 \cdot a_2)b_1) & \text{if } i = 2 \\ b_1 \times b_2 & \text{if } i = 3 \end{cases} \end{bmatrix}^T \tag{2.13}$$

Here $N(\cdot)$ is the normalization function. It was proved by the sanity tests introduced in the paper that this kind of representation is an efficient way for the training in the deep neural networks, compared with quaternions and Euler angles that are not continuous and have singularities.

2.1.3. Applications

6 DoF pose estimation is a central technology that can be the critical part of many computer vision applications such as augmented reality(AR), robotics, 3D scene understanding and autonomous driving.

Augmented Reality

AR applications use 6 DoF pose estimation to accurately place the virtual objects in the real world. With precise estimation and quick inference of the pose guarantee a immersive and interactive experience which is the direction of the development of AR applications[9]. Furthermore, 6 DoF pose estimation can also be utilized to track the real world objects, enabling more natural interactions.

Robotics

6 DoF pose estimation helps robots to understand the scene so that the grasping and manipulation of objects can be achieved. In the field of medical robotics, it can be used to track the surgical instrument or a patient's body part[10]. In manufacturing, robots use the estimated pose to identify, sort and assemble the objects in field like automatic logistic sorting and manufacturing line.

3D Scene Understanding

In order to register the 3D objects into the scene or reconstruct the 3D environment from 2D images or 3D point clouds, 6 DoF pose estimation is required. The alignment of the 3D objects or 3D scenes is realized by estimating the rigid transformation using method like correspondance matching[11] or direct transformation estimation[12] follows the ideas of ICP[13].

Autonomous Driving

Autonomous driving is also a cross-domain topic that requires many different technologies to work together. A well estimated pose of the vehicle inside the scene is the basis of many other subtasks of autonomous driving such as collision avoidance, trajectory planning and so on. Subtle errors in the pose estimation may lead to fatal consequences[14], because the vehicle move normally in high speed and the heading direction cause a large deviation in a long distance considering also the reaction time of the vehicle.

2.1.4. Challenges

6 DoF pose is widely used in many applications and became a popular research topic of computer vision in recent years. However, solving this problem is not trivial and even challenging in many cases.

First constrain would be the auto-occlusion or symmetries of the object since the object cannot be clearly and unequivocally observed from all angles[15]. The auto-occlusion means that the object itself is partially occluded by other parts of the object such as LINEMOD-O dataset[16]. This is common in many real world objects such as table or chair. The symmetries of the object means that the object has same appearance from different angles, which will cause ambiguity in the estimation such as T-LESS dataset[17]. Imagining an image of mug with the handle hidden behind it, it is hard to tell the orientation of the mug without the handle.

Textureless object is also a challenge for 6 DoF pose estimation, since many methods rely not only on the geometry of the object but also on the texture. It is hard for RGB-only methods[18] or keypoint based method[19] to extract enough local features if the object is complete textureless.

Another difficulty is the domain gap between the training and testing data. Normally, the training data consists of synthetic CAD models and images which are clean and annotated with the ground truth pose in order to have a precise supervision. But lacking the information of the real world, for example lighting and occlusion, the model trained on the synthetic data cannot generalize well to the real world data. Some dataset provides the real world data or 3D rendered images which can reduce the domain gap in some degree[20], but the noise and unvalid training samples still confuse the model.

If facing the multi-object scenario, which is common in the application like robotics and autonomous driving, the unknown number and type of objects will increase the difficulty of pose estimation for each object in the scene.

——————image here——————-

2.2. Diffusion Model

2.2.1. Generative Models

One of the most fascinating and distinctive feature of human brain is the ability to create or imagine objects that do not immediately exist in reality. Humans can spontaneouesly

learn the underlying properties of the world and generate the hypotheses of the future. This procedure is similiar to supervised learning and reinforcement learning with little mount of labeled data, but generalizes very well to many unseen scenarios and has a high level of robustness[21].

In order to achieve the similiar ability of the generative process from the human brain, many generative models have been proposed in recent years, to not really synthesize the unseen data but to recover or modifiy the seen data with given constraints. Some of the most popular generative models are introduced below.

Generative Adversarial Networks

Generative Adversarial Networks(GANs)[22] is a smart idea to train a generative model by playing a min-max game between two neural networks. The generator G is trained to generate data that is indistinguishable from the real data, while the discriminator D is trained to distinguish the real data from the fake data generated by G. The training process can be formulated as the value function $V(D, G)$, and for the classification objective using cross entropy loss, the optimization problem can be written as

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.14}$$

The generator is optimized to maximize the probability of that the discriminator will classify the generated data as real, which explains the word "adversarial" in the name of GANs.

GANs have shown a great success in many applications such as generating high-resolution images which are difficult to distinguish from the real ones and the ability to learn the complicated distributions. However, the main challenge of GANs is the instablitity of the training process, which increases the difficulty of training and tuning the model. It will sometimes suffer from the mode collapse problem, where the generator only learns to generate a subset of the data distribution[23].

Some works have been done to solve these problems. For example, Wasserstein GANs[24] use a different loss function to stablize the training process and avoid the mode collapse. Spectral Normalization[25] is another method to stablize the training process by constraining the Lipschitz constant of the discriminator.

Variational Autoencoders

Variational autoencoders(VAEs)[26] is another popular generative model that is based on the encoder-decoder architecture. It allows the model to learn the latent representation of the input data and generate new data from the latent space. The encoder E is trained to map the input data x to the latent space z with a distribution $q(z|x)$, while the decoder D is trained to reconstruct the input data from the latent space z with a distribution $p(x|z)$. The training process can be formulated as

$$\min_{E,D} \mathbb{E}_{x \sim p_{data}(x)}[\mathbb{E}_{z \sim q(z|x)}[\log p(x|z)]] - KL(q(z|x)||p(z)) \tag{2.15}$$

The first term is the reconstruction loss, which is the negative log-likelihood of the input data x given the latent representation z. The second term is the regularization term, which is the

Kullback-Leibler divergence between the latent distribution $q(z|x)$ and the prior distribution $p(z)$. With the regularisation term, we prevent the model to encode the input data far apart in the latent space, which will cause the model to generate unrealistic data.

And the reparameterization trick[27] is introduced afterwards to make the stochastic part of the loss function which is the latent representation z differentiable, so that the model can be trained with backpropagation. The latent representation z is sampled from a distribution $q(z|x)$, which is normally a Gaussian distribution. The trick constructs the random variable z into following expression where ϵ is a random variable sampled from a standard Gaussian distribution.

$$z \in \mathcal{N}(\mu, \sigma^2) \longrightarrow z = \mu + \sigma \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, 1) \tag{2.16}$$

VAEs allow us to easily sample the latent representation z from the prior distribution $p(z)$ and generate novel data from the decoder D. It can alse be used to make data compression and denoising, which is the main application of autoencoders. Since the flexibility and the robustness of VAEs, It is widely used in many applications such as image manipulation, text generation and speech synthesis.

Normalizing Flows

Normalizing flows[28] are a familiy of generative models with tractable marginal likelihood which can not be achieved with VAEs. A normalizing flow is a transformation of a simple distribution into a more complex distribution by a series of invertible and differentiable mappings. By repeating the rule of transformation, the initial probability densitiy "flows" through the sequence of invertible mappings and become a valid distribution.

The basic rule for transformation of densities considers an inverible, smooth mapping $f : \mathbb{R}^D \rightarrow \mathbb{R}^D$, with inverse $f^{-1} = g$. Transforming a random variable z with distribution $q(z)$ through f results in a random variable $z^{'} = f(z)$ has a distribution:

$$q(z^{'}) = q(z) \left| \det \frac{\partial f^{-1}}{\partial z^{'}} \right| = q(z) \left| \det \frac{\partial f}{\partial z} \right|^{-1} \tag{2.17}$$

The last term is the Jacobian determinant of the transformation f, which is the determinant of the matrix of partial derivatives of f with respect to z. Given a chain of invertible mappings $f_1, ..., f_K$, the transformation of the random variable z through the sequence of mappings and the density $q_K(z)$ can be written as

$$z_K = f_K \cdot ...f_2 \cdot f_1(z_0) \tag{2.18}$$

$$lnq_K(z_K) = lnq_0(z_0) - \sum_{k=1}^{K} ln \left| \det \frac{\partial f_k}{\partial z_{k-1}} \right| \tag{2.19}$$

The path of the transformation can be seen as a flow of the probability density from the initial distribution $q_0(z_0)$ to the final distribution $q_K(z_K)$. If the length of the normalizing flow tends to infinity, the model becomes an infinitesimal flow which is described by a differential equation.

Normalizating flows provide a flexible framework for modeling complex distributions, which is difficult to achieve with previous generative models. However the samples that are generated througn flow-based models are not as realistic as the samples from GANs or VAEs, and the data will be projected into also high dimensional space, which is hard to interpret.

Diffusion Model

Diffusion models are a new class of state-of-the-art generative models that can synthesize high-quality images in recent years. The representative one, which is the Denosing Diffusion Probabilistic Models(DDPM) was initialized by Sohl-Dickstein et al[29] and proposed recently by Ho. et al[30].

A diffusion probabilistic model(diffusion model), inspired by the nonequilibrium thermodynamics, is a parameterized Markov chain trained using variational inference to produce samples from a given target distribution after finite steps. The basic idea behind diffusion models is trivial. Given an input data x_0, we first gradually add Gaussian noise to it and finally get a sequence of noised data $x_1, ..., x_T$, which we call it forward process. Afterward, a neural network is trained to recover the original data by estimating the noise and reversing the forward precess, which we call it sampling process or reverse process.

Figure 2.2 shows an overview of four different types of generative models that we mentioned in this section. Unlike VAE and flow-based model, diffusion model is based on the Markov chain and the latent variable has high dimensionality which has the same size of the input data. We recover the original data by estimating the noise and reversing the forward process iteratively, which ensures the high quality of the generated data.

The great succuss of some architecture using the diffusion model such as GLIDE[31] and DALLE-2/3[32] has shown the potential of the diffusion model in the field of generative models. The advantage of diffision model is that it is large-scale, flexible and offer high-quality samples. With the tradeoff of the relative longer training time and inference time because of its 2-phases architecture, it can synthesize highest-quality images than other generative models. This potential motivates us to apply the diffusion model also to 3D domain and the related tasks.

2.2.2. Theory and Fundamentals

In this section, we will introduce the detail of the diffusion model, the mathematical background in the forward process and the sampling process, and the conditional diffusion model. The extended version of the classic DDPM will also be briefly introduced.

Forward Process

Given an input data $\mathbf{x}_0$ from the target data distribution $q(\mathbf{x})$, we first define a forward process that gradually adds Gaussian noise to $\mathbf{x}_0$ with variance $\beta_t \in (0, 1)$ at each step t and finally get a sequence of noised data $\mathbf{x}_1, ..., \mathbf{x}_T$. At each step t, we have the new data x_t with the conditional distribution $q(x_t|x_{t-1})$ defined as:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{2.20}$$

where $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ is a normal distribution with mean $\sqrt{1-\beta_t}\mathbf{x}_{t-1}$ and variance $\beta_t\mathbf{I}$. Thus, we can derive the posterior distribution from the input data $\mathbf{x}_0$ to $\mathbf{x}_T$ in a tractable way:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \tag{2.21}$$

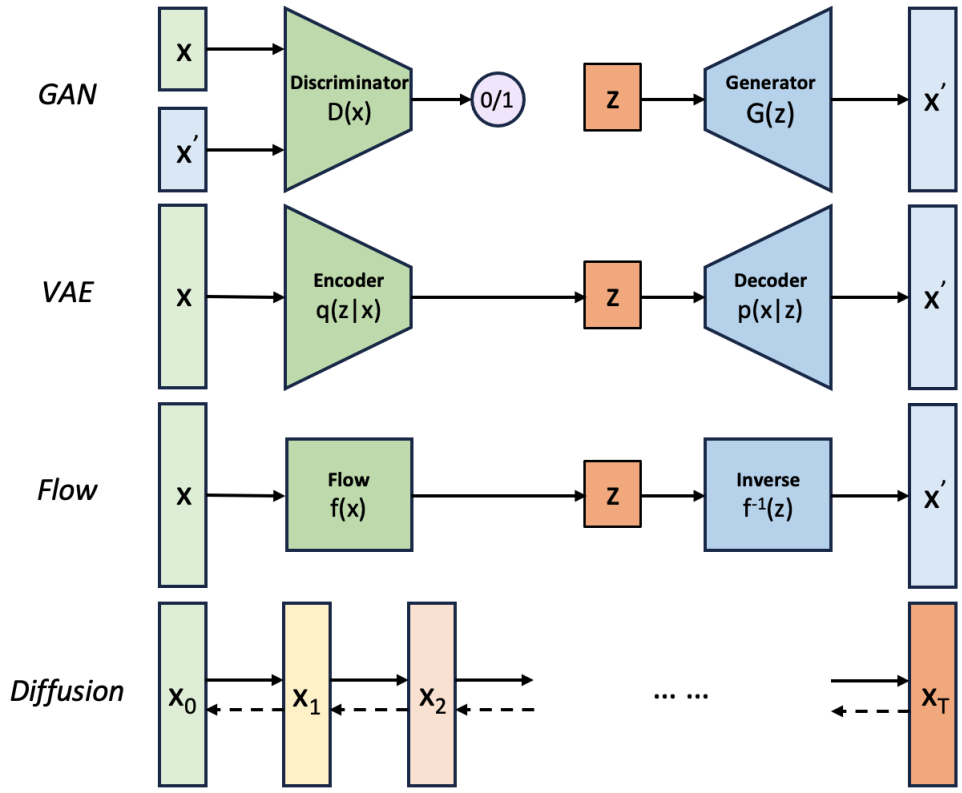

Figure 2.2.: Overwiew of different types of generative models

Our goal is to track the noised data at an arbitrary step t with a close-form posterior distribution $q(\mathbf{x}_t|\mathbf{x}_0)$. So the reparemeterization trick is introduced so that we don't need to calculate the $\mathbf{x}_t$ iteratively from $t = 0$.

Let $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$ with Gaussian noise $\boldsymbol{\epsilon}_0, ..., \boldsymbol{\epsilon}_{t-2}, \boldsymbol{\epsilon}_{t-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, we can simplify the noised the data $\mathbf{x}_t$ in such a recursive way:

$$\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} \\
&= \sqrt{\alpha_t}(\sqrt{\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_{t-1}}\boldsymbol{\epsilon}_{t-2}) + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} \\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\boldsymbol{\epsilon}_{t-2} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1}
\end{aligned}$$

Notice that when we merge two Gaussian distributions with different variance, $\mathcal{N}(\mathbf{0}, \sigma_1^2\mathbf{I})$ and $\mathcal{N}(\mathbf{0}, \sigma_2^2\mathbf{I})$, the new merged distribution is $\mathcal{N}(\mathbf{0}, (\sigma_1^2 + \sigma_2^2)\mathbf{I})$. So we can merge the second and third term in the equation above where $\bar{\boldsymbol{\epsilon}}_{t-2}$ is the new Gaussian and get:

$$\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\boldsymbol{\epsilon}_{t-2} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} \\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1}) + (1-\alpha_t)}\bar{\boldsymbol{\epsilon}}_{t-2} \\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{\boldsymbol{\epsilon}}_{t-2} \\
&= ... \\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}
\end{aligned} \tag{2.22}$$

Finally, we can represent the sample $\mathbf{x}_t$ with the following distribution:

$$\mathbf{x}_t \sim q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t;\ \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{2.23}$$

where α_t and $\bar{\alpha}_t$ can be precomputed for any arbitrary step t from β_t. The variance hyperparemeter β_t is normally chosen as a linear, quadratic or cosine schedule. The orginal design of DDPM used a linear schedule from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$ which is also commonly used in other diffusion models.

Reverse Process

The purpose of the reverse process is to reverse the forward process above and recover the original data $\mathbf{x}_0$ from a random Gaussian noise $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Practically, the reverse conditional distribution is not directly tractable, because the computations involve the whole data distribtion. Therefore, we need to train a model $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ to estimate the reverse conditional distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$. Since the variance β_t is small enough, $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ can be treated as Gaussian distribution, so does $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$, which can be defined as follow:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \tag{2.24}$$

Applying the estimated reverse conditional distribution for all timesteps we get:

$$p_\theta(\mathbf{x}_{0:T}) = p_\theta(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \tag{2.25}$$

The reverse conditional probability is only trackable when conditioned on $\mathbf{x}_0$:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t\mathbf{I}) \tag{2.26}$$

With the help of Bayes' Rule and the properties of Gaussian probability density function, we can prove that:

$$\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t \tag{2.27}$$

$$\tilde{\boldsymbol{\mu}}_t = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x_t} + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x_0} \tag{2.28}$$

Thanks to the reparemeterization trick, we can represent $\mathbf{x_0} = \frac{1}{\sqrt{\alpha_t}}(\mathbf{x_t} - \sqrt{1-\bar{\alpha}_t}\epsilon_t)$ from 2.22 and further simplify the expression of $\tilde{\boldsymbol{\mu}}$ as:

$$\tilde{\boldsymbol{\mu}}_t = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_t\right) \tag{2.29}$$

Notice that such a setup of p and q is similar to VAEs, so we can optimize the negative log-likelihood using the variational bound:

$$\begin{aligned}
-\log p_\theta(\mathbf{x}_0) &\leq -\log p_\theta(\mathbf{x}_0) + D_{KL}(q(\mathbf{x}_{1:T}|\mathbf{x}_0)||p_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0)) \\
&= -\log p_\theta(\mathbf{x}_0) + \mathbb{E}_q\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})/p_\theta(\mathbf{x}_0)}\right] \\
&= -\log p_\theta(\mathbf{x}_0) + \mathbb{E}_q\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})} + \log p_\theta(\mathbf{x}_0)\right] \\
&= \mathbb{E}_q\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] =: L
\end{aligned} \tag{2.30}$$

To make the lower bound L computable, the expression can be further rewritten after some manipulations in Appendix of [30] as:

$$L = \mathbb{E}_q \left[\underbrace{D_{KL}(q(\mathbf{x}_T|\mathbf{x}_0)||p_\theta(\mathbf{x}_T))}_{L_T} + \sum_{t=2}^{T} \underbrace{D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{- \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0} \right] \tag{2.31}$$

Each term L_i with $i \in \{0, ..., T\}$ compares the forward and reverse conditional distributions at each timestep i and in closed form, where L_T is constant and can be ignored during training, L_0 is the reconstruction term and is learned using a seperate decoder in the original model[33].

The second term L_{t-1} describe the difference of $p_\theta(\mathbf{x}_t|\mathbf{x}_{t-1})$ against the posteriors in forward process, which we need to learn during the training process. Replace $t-1$ with t and t with $t+1$ in the equation above in order to express it in a natual way, we use L_t in the following calculation.

Revisit the reverse process from 2.24, we need to train $\boldsymbol{\mu}_\theta$ to approximate $\tilde{\boldsymbol{\mu}}_t$ in 2.29, where ϵ_t can be reparameterized as the prediction from the input $\mathbf{x}_t$ at time step t. Finally, we can have the expression of the approximation of the mean:

$$\boldsymbol{\mu}_\theta = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) \tag{2.32}$$

The lost term L_t can be formulated using l_2 distance:

$$L_t = \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}} \left[\frac{1}{2\,||\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)||_2^2} \left\| \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t, t) \right\|^2 \right] \tag{2.33}$$

which can be simplified ignoring the weighting term according to the original paper[30] as:

$$L_{simple} = \mathbb{E}_{t\sim[1,T],\mathbf{x}_0,\boldsymbol{\epsilon}} \left[\left\| \boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t, t) \right\|^2 \right] + C \tag{2.34}$$

where C is a constant term which not related to θ and can be ignored during training. And we the variance is not considered in the loss function and it is improved in the later research[34] to let the network also learn the covariance matrix $\boldsymbol{\Sigma}_\theta$.

An overview of the forward and reverse process using a cat image as example is shown in figure 2.3.

Conditional Diffusion Model

Conditional diffusion, also called guided diffusion is very practical in many applications since we normally want to generate the data in paricular style, direction or distribution and not in an arbitrary way.Typical usage of conditional diffusion is to sample data from a given class or category, as well as text prompt, image prompt and so on.

Mathematically, condition means the prior distribution $p(\mathbf{x})$ is conditioned on a given input y. By modifiying the equation 2.25, we get

$$p_\theta(\mathbf{x}_{0:T}|y) = p_\theta(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, y) \tag{2.35}$$

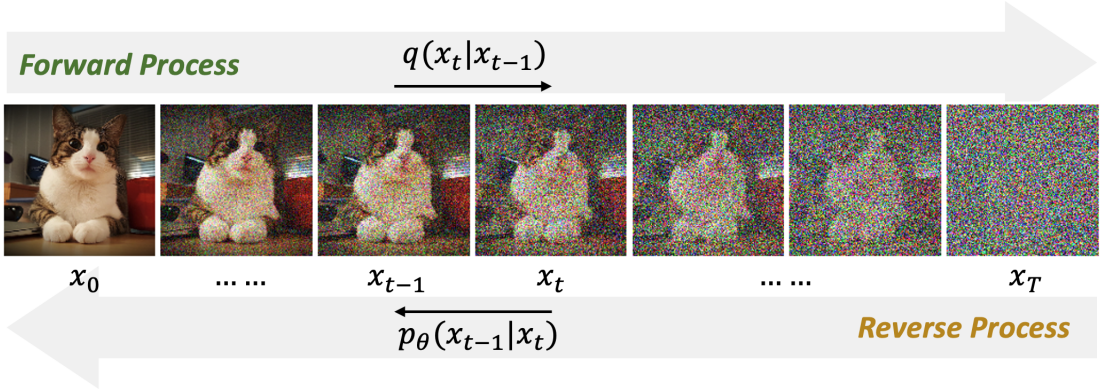

Figure 2.3.: Forward and reverse process of diffusion model using 2D image as example

Using the idea of the score-based generative model[35], we can train a score network for an unconditioned diffusion with score function:

$$\mathbf{s}_\theta(\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t) \tag{2.36}$$

Extend the score function with condition y, we can get the conditional score function after applying Bayes' Rule:

$$\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t|y) = \nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t) + \nabla_{\mathbf{x}_t} \log p_\theta(y|\mathbf{x}_t) \tag{2.37}$$

Based on the score function we can derive the conditional diffusion model with two variations, namely the classifier guidance and classifier-free guidance.

Classifier guidance is a method that balances the trade-off between mode coverage and sample fidelity post-training. It combines the score estimate of a diffusion model with the gradient of an image classifier, which requires training an classifier f_ϕ separate from the diffusion model and use the gradients of the classifier as the guidance.

Without the an separate classifier f_ϕ, it is still possible to let the conditional and unconditional score function share the same network, which is called classifier-free guidance. The diffusion model is trained by randomly dropping the condition y during training. And the result turns out to be that the conditonal and unconditional score estimates are combined to attain a good tradeoff between quality and diversity[36].

The training and sampling algorithm of the conditional denoising diffusion probabilistic model can be summarized as following algorithm 1 and 2.

Algorithm 1 Training of Conditional Diffusion Model

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

Algorithm 2 Sampling of Conditional Diffusion Model

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

Extensions

————————Add if needed————————-

2.2.3. Applications

Computer Vision

The majority of the applications of diffusion models lies in the field of computer vision, including super resolution, translation, inpainting and so on[37]. Diffusion models have shown a great performance in these 2D based manipulation tasks compared with other generative models such as GANs and VAEs.

Super-Resolution via Repeated Refinement (SR3)[38] and Cascaded Diffusion Models (CDM)[39] are two representative works in the field of super resolution. They use either an iterative way or concatination of diffusion models to generate high-resolution image from low-resolution imput. Figure 2.4 illustrates the process of cascaded super resolution, which is taken from the project page of the SR3. Implicit Diffusion Models (IDM) for Continuous Super-Resolution[40] integrates an implicit neural representation in the decoding process.

Inpainting and image translation are also two popular image manipulation tasks with different conditional inputs. Typical works are RePaint[41], Palette[42] and Diffusion-based Image Translation using Disentangled Style and Content Representation[43].

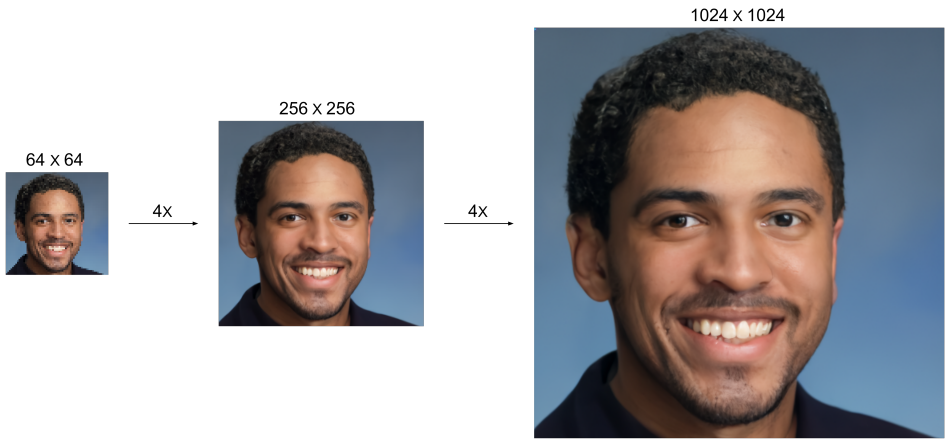

Figure 2.4.: Super resolution using diffusion model

In 3D domain, the diffusion model is also applied to the task of point cloud generation and completion. Luo et al.2021[44] and Zeng et al.2022[45] have presented the diffusion models for point cloud generation by treating point clouds as particles in a thermodynamic system. Lyu et al.2022[46] have proposed a coarse-to-fine point cloud completion diffusion model and also established a point-wise mapping between the output and ground truth. Figure 2.5 adapted from the paper shows the point cloud generation using diffusion architecture.

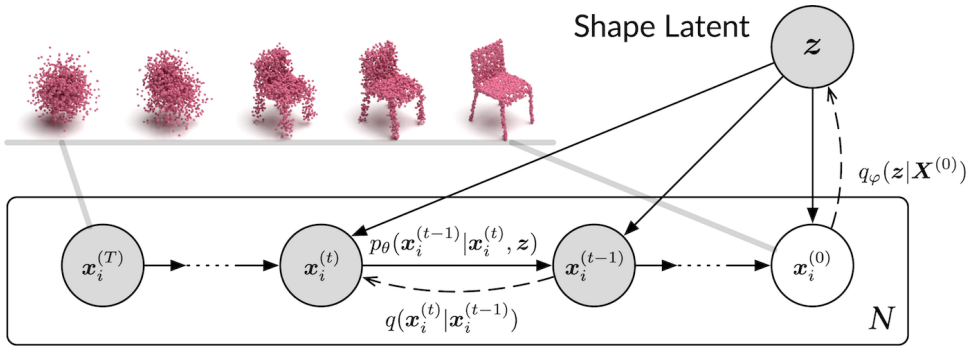

Figure 2.5.: Point cloud generation using diffusion model

Natural Language Processing

Natural language processing (NLP) has been dramatically developed in recent years. The iconic models like BERT[47], GPT series[48] and LLaMA[49] are all based on the transformer architecture. However, there are also some diffusion based methods that can generate text with high quality and diversity. In fact, diffusion models have been shown to have significant advantages over autoregressive models in terms of parallel generation, text interpolation, token-level controls such as syntactic structures and semantic contents, and robustness[50].

Discrete Denoising Diffusion Probabilistic Models (D3PMs)[51], is diffusion-like generative models for discrete data that generalize the multinomial diffusion model[52], by going beyond corruption processes with uniform transition probabilities.

Diffusion-LM[53] proposes a non-autoregressive language model based on continuous diffusions, which iteratively denoises a sequence of Gaussian vectors into word vectors, yielding a sequence of intermediate latent variables, which makes it possible for simple, gradient-based methods to achieve complex control.

Multi-Modal Learning

Multi-modal learning is a field that combines different modalities such as text, image, video and audio. It tends to become the mainstream of the future research in the field o machine learning because of the higher requirement of the real-world applications.

Text-to-Image generation is a typical task in this field. A common pipeline is to first train a prior model that can generate image embedding conditioned on a text prompt,e.g. CLIP[54]. Then we use the prior output as condition to train a diffusion model to generate the final image. Famous works like Stable Diffusion[55] and DALLE-2[32] followed this pipeline

and achieved state-of-the-art results in text-to-image generation. Following Text-to-Image samples 2.6 are generated using Stable Diffusion.

Figure 2.6.: Image synthesis with text prompt in different styles using Stable Diffusion

ControlNet[56] attempts to control pre-trained large diffusion models to support additional semantic maps, like edge maps, segmentation maps, keypoints, shape normals, depths, etc. Authors use the "trainable copy" of the original weights of the pretrained diffusion model and connect these "copy" blocks with the original model with zero convolution layer. Thus, we don't need to retrain the whole model and also gaurentee the quality as well as the flexibility of the model.

Text-to-3D generation and Image-to-3D are novel tasks in the field of multi-modal learning and has the potential to be applied in many cases such as 3D object reconstruction, 3D scene generation and so on. DreamFusion[57] adopts a pre-trained 2D text-to-image diffusion model to perform text-to-3D synthesis. It optimizes a randomly-initialized 3D model (a Neural Radiance Field, or NeRF) with a probability density distillation loss, which utilizes a 2D diffusion model as a prior for optimization of a parametric image generator. Figure 2.7 illustrates the process of the text-to-3D generation using DreamFusion, and the image is adapted from the demo in original work.

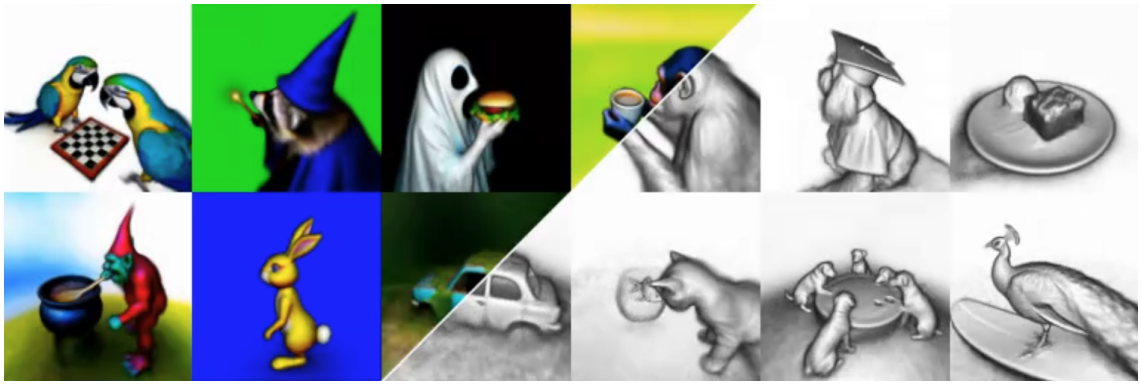

Figure 2.7.: 3D model (NeRF) synthesis prompt using DreamFusion

3. Related Work

4. Pose Hypotheses Diffusion

4.1. Introduction

In this chapter, we will introduce the detail of the first proposed method, namely the pose hypotheses diffusion. The intuitive idea is to directly denoise the pose which consists of translation and rotation, $\mathbf{T} = (\mathbf{t}, \mathbf{r})^T$. As the diffusion model is basically one of the generative models, we use the diffusion pipeline to generate the possible pose hypotheses which is similar to the image synthesis but with different objective. So we call this kind of pose estimation method as pose hypotheses diffusion.

Through the experiments using different representation of the rotation, we find that the 6D representation of the rotation introduced in 2.11 is the most efficient one. So we use this representation in the following chapters and the comparision with other forms of rotation will be discussed in the experiment section.

Repeating the sampling process of the pose for one reference input $\mathbf{r}$, we can get a set of pose hypotheses $\mathbf{T}_1, ..., \mathbf{T}_N$, which compose a hypothesis distribution $h(\mathbf{T}|\mathbf{r})$ that can be used to estimate the pose $\mathbf{T}$ of the reference input $\mathbf{r}$. For a given object without any ambiguity, the distribution $h(\mathbf{T}|\mathbf{r})$ should be a delta distribution in ideal case, or in other words, sqeezed to a single point in the spatial solving space. The pose hypotheses $\mathbf{T}_1, ..., \mathbf{T}_N$ should be close to each other and the variance of the distribution should be small. On the contrary, if the object is symmetrical, the distribution of the pose should fit the corresponding pattern of the symmetry in the solving space.

————————image here————————-

4.2. Methodology

This section introduces the overall structure of the pose hypotheses diffusion, including the 2-phase architecture of the diffusion model and the other modules in the whole model.

4.2.1. Structure

Similarly to the original diffusion pipeline, we first need to train a model to estimate the noise $\boldsymbol{\epsilon}_\theta$ conditioned with the input $\mathbf{x}_t$, the timestep t as well as the guidance $\mathbf{y}$. Then we can go through the reverse process in which we generate the pose hypothesis $\mathbf{T}_{t-1}$ with conditional distribution $q(\mathbf{T}_{t-1}|\mathbf{T}_t, \mathbf{y})$ step by step using the equations we derived in section 2.2.2.

Training Phase

In the training phase of the pose hypotheses diffusion, we basically let the backbone network to predict the noise given by the noised pose $\mathbf{T}_t$, the noised pose can be derived from the reference pose $\mathbf{T}_0$ with the noise schedule β_t which introduced before in the forward process. The most application using diffusion has a convolutional UNet-like backbone[58], which performs well in the 2D tasks. However, dealing with the pose estimation task, we have a different objective and convolutional neural network is no longer suitable. In our model, we utilize the transformer encoder as the backbone network, which has be proved to be effective and flexiable in not only the natual language processing but also the computer vision tasks.

Additionally, we also need to provide the timestep t and the guidance $\mathbf{y}$ to the backbone. The guidance here is the feature of the reference RGB or RGB-D image depending on the requirement of the task or the dataset. We use RGB-D image in our experiments which has both 2D and 3D features can be extracted and fused in to the model. As 2D feature extractor, a pretrained self-supervised Vision Transformer with DINO[59] is used and the 3D feature is extracted from a pretrained FoldingNet encoder[60]. The structure of the training process is shown in figure 4.1.

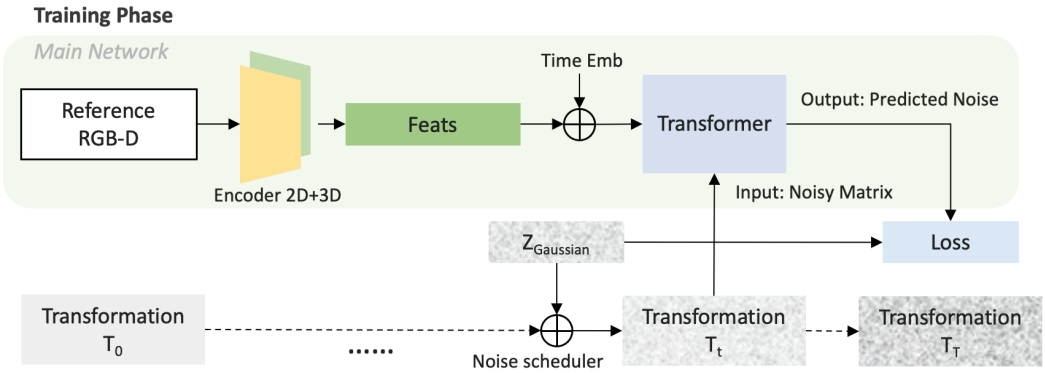

Figure 4.1.: Structure of the training phase of the pose hypotheses diffusion

————————algo————————

Sampling Phase

Assuming that the denoiser is converged in the training phase, we can use the denoiser to iteratively generate the pose hypotheses with the randomly initialized the transformation $\mathbf{T}_T$. Same as the training phase, we need to provide the timestep t and the guidance $\mathbf{y}$ to the backbone. Given a reference RGB-D image, we use the pretrained 2D and 3D encoder to extract the downstream features and concatinate them as the conditional embedding of the diffusion backbone.

Since during the training phase, the network has learned the noise distribution conditioned each timestep embedding, so the denoiser has the capability to predict the noise $\boldsymbol{\epsilon}_\theta$ from $t = T$ to $t = 0$. Then we can use the equation 2.24 to get the pose hypothesis $\mathbf{T}_{t-1}$ and repeat the process until we get the final pose hypothesis $\mathbf{T}_0$.

In the training phase, we feed the network with batch of data that is different in the timestep t and the guidance $\mathbf{y}$ (different reference images). During the sampling phase, we can easily make the batch size to one and only infer one pose hypothesis $\mathbf{T}_0$ for one reference input. Another efficient way is to simultaneously sample multiple pose hypotheses by batchifying the randomly initialized transformation and conditioned with the same reference input. And the mean of the sampled pose hypotheses can be more precisely estimated as the final pose estimation if the pose of the object is uniquely determined in the space. This can be switched that the batch of the random Gaussian initialization is conditioned with different reference inputs, which can infer multiple pose for different frames or different objects.

After the optional multi-hypotheses inference, we can further use some algorithms to refine the pose. In our model, we choose iterative closest point (ICP)[61] algorithm to refine the pose hypotheses and finally get the output transformation $\mathbf{T}_r$. The structure of the sampling process is shown in figure 4.2. Details of each models in training phase and sampling phase will be introduced in next section.

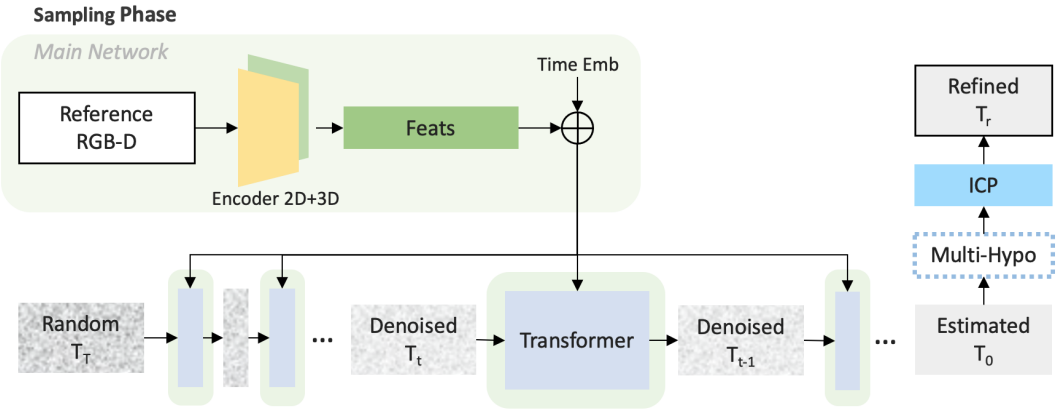

Figure 4.2.: Structure of the sampling phase of the pose hypotheses diffusion

Speed up Sampling with DDIM

It is relatively slow to generate the sample from DDPM, because the denoising process follows the whole Markov chain of the reverse process which is quite long. Slow inference reduced performance of the real-time application of pose estimation, which we need to optimizd. the In order to speed up the sampling process, we use the dinoising diffusion implicit model (DDIM)[62] with same training procedure as DDPM but more efficient in the reverse process.

First, we rewrite the reverse conditional probability in 2.26 to be parameterized by a desired standard deviation σ_t using the reparameterization trick in 2.22 as:

$$q_\sigma(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1};\ \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1-\bar{\alpha}_t}}, \sigma_t^2\mathbf{I}) \tag{4.1}$$

Compared with the original probability $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t\mathbf{I})$, we have:

$$\tilde{\beta}_t = \sigma_t^2 = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t \tag{4.2}$$

Then we let $\sigma_t^2 = \eta \cdot \tilde{\beta}_t$ where η is a hyperparemeter that controls the sampling stochasticity. $\eta = 0$ corresponds the best performance of the model and $\eta = 1$ corresponds to the original DDPM. In the reverse process, we only sample a subset of the complete diffusion steps $\{\tau_1, ...\tau_S\}$ and the reverse process can be formulated as:

$$q_{\sigma,\tau}(\mathbf{x}_{\tau_i-1}|\mathbf{x}_{\tau_i}, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{\tau_i-1};\ \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\frac{\mathbf{x}_{\tau_i} - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1-\bar{\alpha}_t}}, \sigma_i^2\mathbf{I}) \tag{4.3}$$

With DDIM we can use a non-Markovian diffusion process whose reverse process can be much faster than the original DDPM, and still keep the quality of the samples. It also remains the consistency of the reverse process when $\eta = 0$, where the reverse process is deterministic. That makes sure that multiple samples from the same reference input have similar high-level features and in our case the pose hypotheses are consistent with each other.

4.2.2. Models

Denoiser Network

The following figure 4.3 illustrates the structure of our main network in diffusion model. As in previous section mentioned, the transformer encoder processes the translation and rotaion vector together with their position embedding conditioned with time embedding and 2D/3D feature embedding of the reference image and predicts the noise added at this timestep.

It is worth mentioning that the fusion of 2D and 3D feature is not pointwise aligned, which means we don't extract the pointwise feature from the 2D and 3D domain and feed to the network. Instead of doing that, we seperately extract the global features and concatinate them together in order to reduce the difficulty of the convergence of the backbone with the tradeoff of the robustness and generalization of the model. This part of optimization will be discussed later.

Backbone

A modified transformer encoder is utilized as the backbone of our diffusion model. The Transformer[63] is a sequence-to-sequence model that uses multi-head self-attension layers to understand the relevant token in the sequence and follows the encoder-decoder structure in the NLP tasks. However, in our case the encoder part is what we need to estimate the noise.

Similar to the vanilla transformer, our model consists of N stacked transformer encoder blocks. As shown in the left part of figure 4.3, each block is made up of two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. We use residual connections around each of the two sub-layers and the Layer norm (LN) is applied before each sub-layer, which follows the design of Vision Transformer (ViT)[64].

The core of the transformer encoder is the multi-head self-attention mechanism, which is illustrated in figure 4.4. First, we create three vectors from each input vector $\mathbf{x}_i$, namely the query vector $\mathbf{q}_i$, the key vector $\mathbf{k}_i$ and the value vector $\mathbf{v}_i$. These vectors are created by multiplying the input vector $\mathbf{x}_i$ with three matrices $\mathbf{W}^Q$, $\mathbf{W}^K$ and $\mathbf{W}^V$ respectively that we

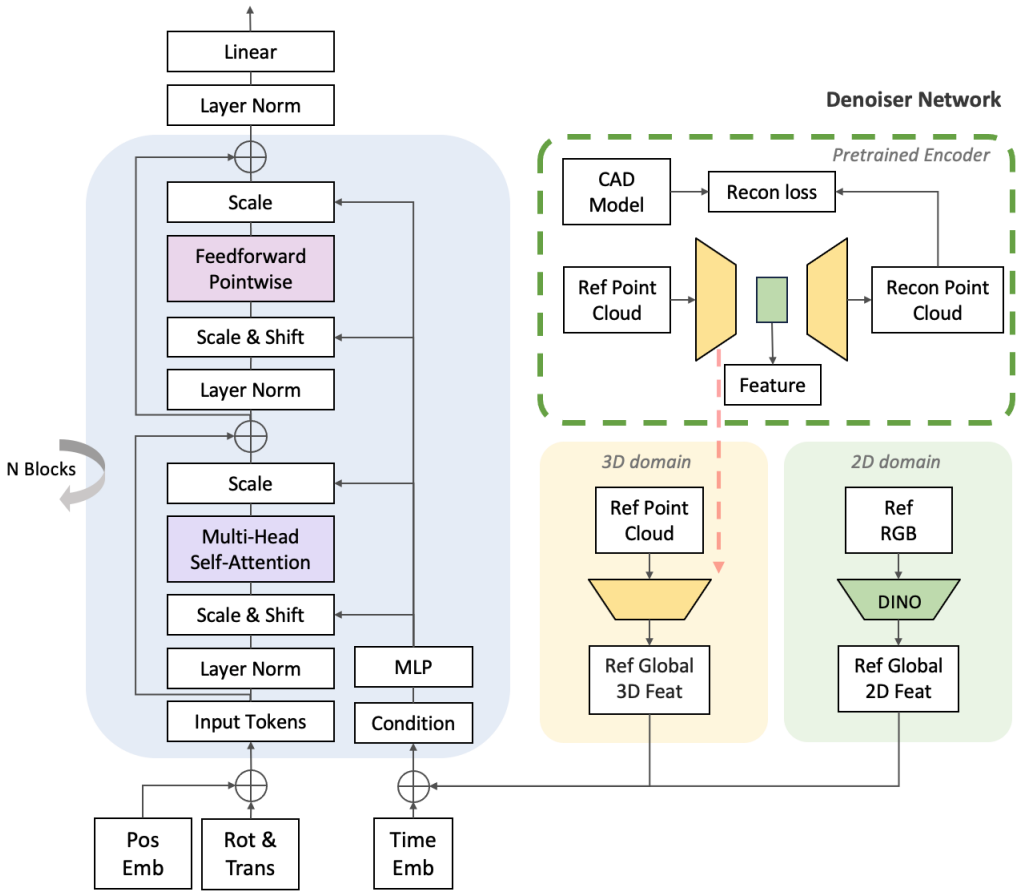

Figure 4.3.: Denoiser network with backbone and feature extractor

trained during the training phase. Then we use the scaled dot-product attention to calculate the output vector $\mathbf{y}_i$:

$$\mathbf{y}_i = \text{softmax}\left(\frac{\mathbf{q}_i \mathbf{k}_i^T}{\sqrt{d_k}}\right)\mathbf{v}_i \tag{4.4}$$

where d_k is the dimension of the key vector $\mathbf{k}_i$. The scaled dot-product attention is the core of the transformer encoder and the multi-head self-attention is the extension of the scaled dot-product attention. The multi-head self-attention is defined as:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\mathbf{h}_1, ..., \mathbf{h}_n)\mathbf{W}^O \tag{4.5}$$

where $\mathbf{h}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V)$ and $\mathbf{W}_i^Q$, $\mathbf{W}_i^K$, $\mathbf{W}_i^V$ and $\mathbf{W}^O$ are the trainable parameters. The multi-head self-attention allows the model to jointly attend to information from different representation subspaces at different positions. With a linear projection at the end, the model is able to learn a more complex function.

To effectively process the conditional input, we use modified adaptive layer normalization (adaLN) to replace the original layer normalization in the transformer encoder which is introduced in [65, 66]. The learnable scale and shift parameters are regressed from the conditional input and applied in each sub-layer of the transformer encoder blocks. The modified

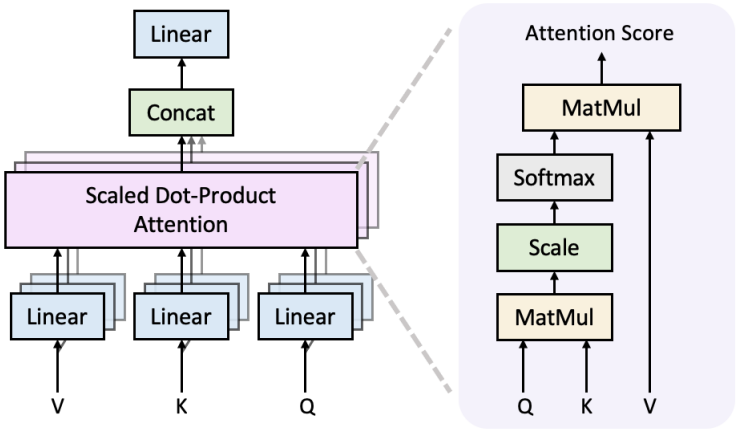

Figure 4.4.: Multi-head self-attention and scaled dot-product attention

transformer encoder block can be formulated as:

$$\mathbf{y}^{'} = \alpha_1(\mathbf{c}) \odot \text{Attention}[\gamma_1(\mathbf{c}) \odot N(\mathbf{x}) + \beta_1(\mathbf{c})] \tag{4.6}$$

$$\mathbf{y} = \alpha_2(\mathbf{c}) \odot \text{FeedForward}[\gamma_2(\mathbf{c}) \odot N(\mathbf{y}^{'}) + \beta_2(\mathbf{c})] \tag{4.7}$$

where $\mathbf{y}$ is the output of the block, $\mathbf{y}^{'}$ is the intermediate expression after the first sub-layer, $\mathbf{c}$ is the conditional input, $\alpha_1(\mathbf{c})$, $\alpha_2(\mathbf{c})$, $\beta_1(\mathbf{c})$, $\beta_2(\mathbf{c})$, $\gamma_1(\mathbf{c})$ and $\gamma_1(\mathbf{c})$ are the learnable parameters and N is the layer normalization. The $\odot$ denotes the element-wise multiplication.

————————Reason for using transformer————————-

Time Embedding and Postion Embedding

Why we need time embedding and position embedding in our case? As the architecture of diffusion model determines that we need to let the network learn the influence of timestep on the noise estimation. So we have to encode the timestep information into the network. For the same reason, the sequence of the transformer input which is the transformation vector is also relevant and should be encoded, because the each bit of the vector represents the different meaning and can not be shuffled.

The way we embed time and position information is generally called positional encoding. It should satisfy the following conditions[67]:

- It should be unique and deterministic defined for each position (or timestep).
- The encoded distance between any two steps should be consistent across different timesteps.
- The value should be bounded and generalize to any input.

The most common positional encoding is the sine and cosine positional encoding[63], which is defined as:

$$\text{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{4.8}$$

$$\text{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{4.9}$$

where pos is the position, i is the dimension and d_{model} is the dimension of the input vector. And we add a fully connected layer to the positional encoding to make it trainable. Figure 4.5 shows the 64-dimensional positional encoding for a sequence with length of 100 using the sine and cosine positional encoding.

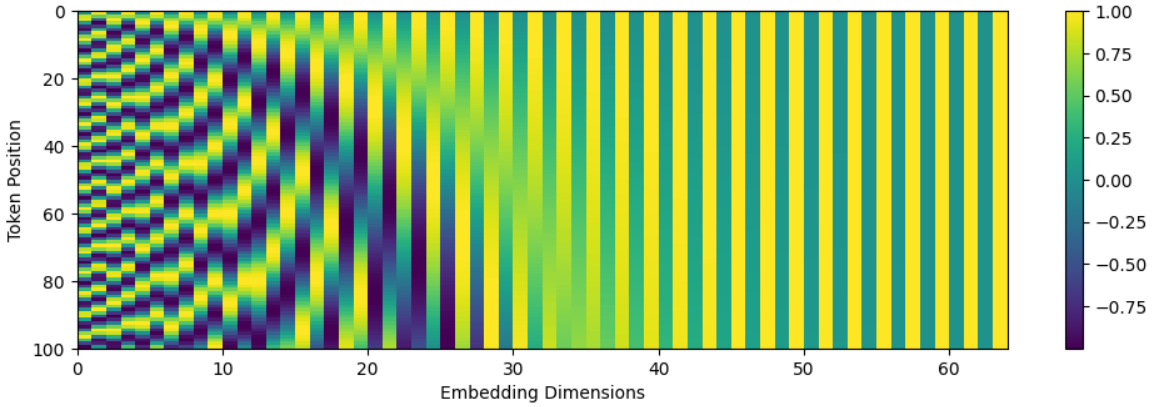

Figure 4.5.: The 64-dimensional positional encoding for a sequence with length of 100

2D Feature Extractor

As the 2D feature extractor, the self-supervised Vision Transformer with DINO[59] is used. DINO is a self-supervised learning method that trains a Vision Transformer with a small set of negative examples. It is a contrastive learning method that maximizes the agreement between differently augmented views of the same image. The architecture of DINO shares the same overall structure with recent self-supervised approaches and also the similarites with knowledge distillation.

Knowledge distillation is a learning paradigm where a student network g_{θ_s} is trained to match the output of a given teacher network g_{θ_t}, parameterized by θ_s and θ_t respectively. Given an input image x, both networks output probability distributions over K dimensions denoted by P_s and P_t. The probability P is obtained by normalizing the output of the network g with a softmax function. Given a fixed teacher network g_{θ_t}, the student network g_{θ_s} is trained to minimize the cross-entropy loss between the two distributions w.r.t the student parameters θ_s:

$$L = \min_{\theta_s} H(P_t(x), P_s(x)) \tag{4.10}$$

where $H(a, b) = -a \log b$ is the cross-entropy loss. And this optimizatiin problem is adapted to self-supervised learning by constructing different distorted views or crops of an image with multi-crop strategy. Making a set of two global views, x_1^g and x_2^g and several local views with smaller resolution, the loss function can be formulated as:

$$L_{DINO} = \min_{\theta_s} \sum_{x \in \{x_1^g, x_2^g\}} \sum_{\substack{x' \in V \\ x' \neq x}} H(P_t(x), P_s(x')) \tag{4.11}$$

The structure of the self-supervised architecture is shown in figure 4.6. The model passes two different random transformations of the input image to both networks with same stucture

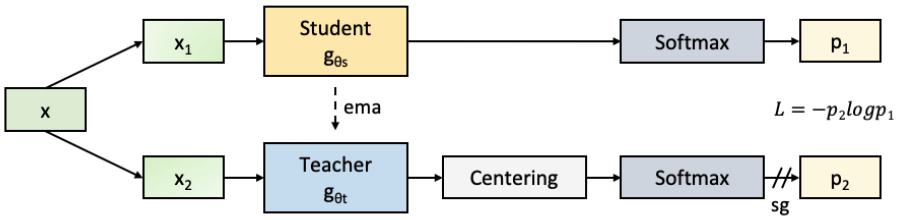

Figure 4.6.: Self-supervised architecture of DINO

but different parameters. The output of the teacher network is centered and a stop-gradient (sg) is applied on the teacher to let the gradients only propagate through the student network. The teacher parameters are updated with an exponential moving average (ema). [h]

The backbone of the DINO is ViT[64], which proved that the transformer architecture also performs well on the 2D vision tasks. The standard transformer processes 1D sequences, and in order to handle 2D images, the input is first flattened into a sequence of patches. For a input image $x \in R^{H\times W\times C}$ and patch size p, the patchified input can be denoted as $x_p \in R^{N\times(p^2C)}$ with the number of patches $N = HW/p^2$. The reason why the patches are fed into the transformer rather the raw image is that it is relatively easier for the network to understand the relationship between the patches than the raw pixels.

Similar to the vanilla transformer for NLP tasks, the patches need to be embedded with the positional encoding. A standard learnable 1D postion embedding is used in the original paper. In order to solve the classification task with ViT, an extra token is add to the sequence of patches, which is called class token. An additional multi-layer perceptron (MLP) layer is used for the classification. But for the downstream tasks like ours, we only need the latent feature from the transformer encoder output. The overview of the ViT model is shown in figure 4.7, using the illustration in the original publication.

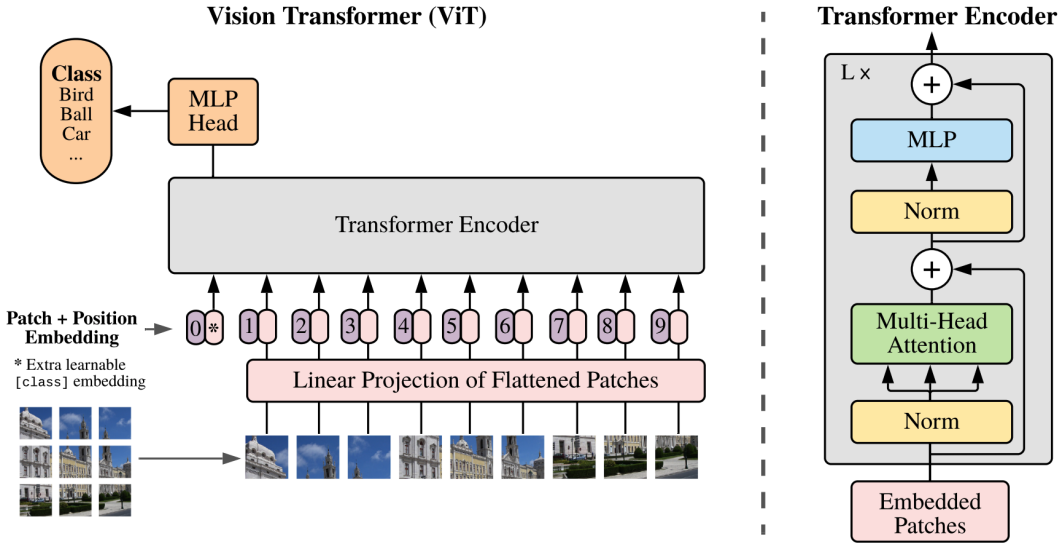

Figure 4.7.: Sturcture of the ViT model

Compared with convolutional neural network which has a dominant position in the field of

2D vision tasks, the Vision Transformer has advantages as well as drawbacks under some scenarios. The transformer is more computationally expensive and requires more memory than the convolutional neural network. And the transformer is not as robust as the convolutional neural network in the case of small dataset. On the other hand, Vision Transformer architecture is more flexible and can be easily applied to different tasks with different input size. It offers a more natural way to process the 2D image, which is more similar to the human brain. And the transformer is more suitable for the tasks that require the global information of the image, such as the classification task. In our case, we use the Vision Transformer pretrained with DINO to extract the global feature of the reference image and feed it to the transformer encoder as the conditional embedding.

We directly use the weights of the model pretrained on ImageNet[68], because the 2D feature downstream network generalizes well on other datasets than 3D feature extractor, which has right now rare well generalized downstream backbone that can cover any 3D objects. Out of this reason, we train the 3D feature extractor from scratch on the dataset we use and this part will be introduced in next section.

3D Feature Extractor

Due to the permutation invariance and transformation invariance of the point cloud, the 3D networks are differently constructed compared with the 2D networks. Famous works like PointNet[69] and PointNet++[70] are the pioneers of the 3D deep learning. After that the convolutional neural network is also introduced to the 3D domain such as KPConv[71].

In our case, we use an encoder-decorder architecture to first build up a point cloud completion task of the target dataset and utilize the latent feature from the encoder output as the 3D global feature of the reference RGB-D image. The reason of using the point cloud completion task rather than classification or semantic segmentation task is because we need the network to learn the geometrical feature such as shape, position and orientation of the object. And the point cloud completion is more suitable than seperately reconstruct the CAD model and reference partial point cloud, because the network can learn the relationship between the two point clouds and the latent feature is more robust and generalizable.

FoldingNet consists of a graph-based encoder and a folding-based decoder. The graph-based encoder follow the design of [72] using the graph structure of the point cloud neighborhood. The encoder is built up with MLP and graph-based maxpooling layers, which are constructed from the k-nearset neighbor through the spatial postion of the nodes in the point cloud. For each point, a local covariance matrix with the size of 3x3 is computed, flattened to a vector and concatinate with the point position as the input vector fed to the network. Afterward, the input vector is passed through MLP and graph layers to get the topology information of the local neighborhood. Then the max-pooling is applied to all nodes and the global feature is projected to a latent space by using another perceptron.

The folding-based decoder is built up with with two consecutive MLP to wrap a fixed 2D grid into the shape of the reference point cloud and that explains why this network is called FoldingNet. First, a 2D grid matrix is initialized in a standard square shape at the origin and is concatenated with the replicated latent feature from the encoder output to become the input of the first folding MLP. The output is the first folded point cloud and it is again concatinated with the replicated latent feature and fed to the second folding MLP.

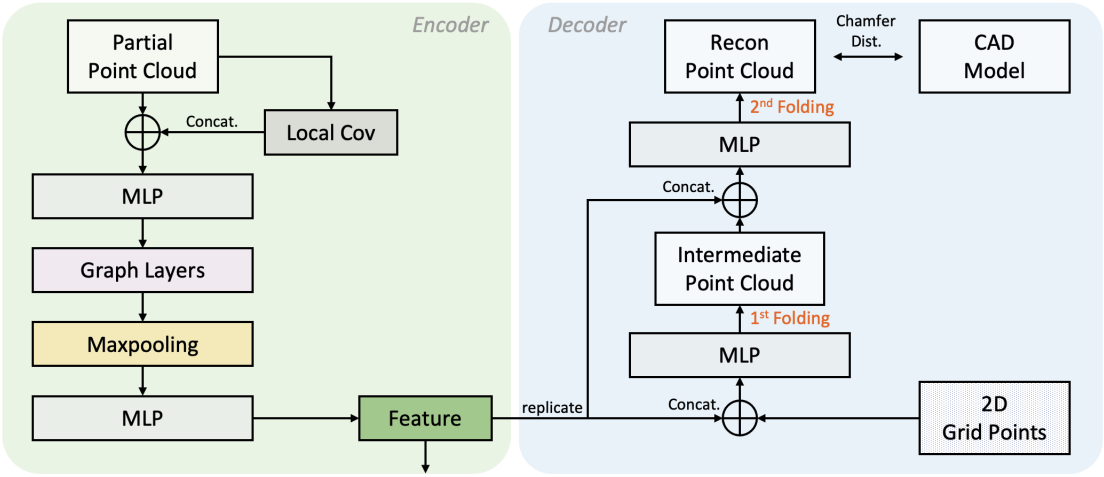

Figure 4.8.: 3D feature extraction with FoldingNet based point cloud completion

The final output point cloud is aligned with the reference point cloud using the bi-directional Chamfer distance as the loss function, which can be formulated as:

$$L_{CD}(S, \hat{S}) = \max \left\{ \frac{1}{|S|} \sum_{x \in S} \min_{\hat{x} \in \hat{S}} \|x - \hat{x}\|_2, \ \frac{1}{|\hat{S}|} \sum_{\hat{x} \in \hat{S}} \min_{x \in S} \|\hat{x} - x\|_2 \right\} \tag{4.12}$$

where S is the reference point cloud and $\hat{S}$ is the output point cloud. The formulation enforces that the output point cloud is close to the reference point cloud and vice versa. In our case, given a partial point cloud as input, the reconstruct point cloud is aligned with the CAD model at the pose of partial point cloud. The structure of the FoldingNet is shown in figure 4.8.

Feature Fusion

Multi-Hypotheses Inference

One of the advantages of diffusion model is the high quality and diversity of the generated objects. Using the case of image synthesis as an example, we can derive dozen of high-resolution and meaningful images from the diffusion model with the same prompt. In our case, we want to explore the diversity of the pose hypotheses if there is ambiguity in the reference input. In another word, for a symmetrical object, we want to derive a distribution which fit the symmetry pattern of the object and for an object with uniquely determined pose, we want to use the multi-hypotheses inference to get a more precise estimation of the pose.
————————————-To be determined————————————-

Pose Refinement

To further refine the 6 DoF pose of the reference input, we can use the coarse pose estimation from the diffusion model as the initial pose and use some algorithms to refine the pose.

Iterative and not learning-based method like random sample consensus (RANSAC)[73] and iterative closest point (ICP)[61] are widely used in the field of pose estimation. RANSAC, for example, uses repeated random sub-sampling to iteratively estimate the optimal result in the set of data containing outliers. The goal of RANSAC is to find a global optimal solution which excludes the outliers and contains the inliers as much as possible. This technique is widly used in the correspondance based point cloud registration. Our pose hypotheses diffusion model samples directly the estimated 6 DoF pose of the reference input and we utilize ICP to the point cloud transformed via the coarse pose estimation.

The ICP algorithm is also an iterative solution to the rigid point cloud registration problem. Given a set of source points A and a set of target points B, the goal of ICP is to find a rigid transformation $(\mathbf{R}, \mathbf{t})$ that minimizes the distance between the source points and the transformed target points. First step is to find the corresponding points pair using nearst neighbor search. Let the set of corresponding pairs be:

$$C = \{(i, j) \mid \mathbf{a}_i \in A \text{ and } \mathbf{b}_j \in B \text{ are corresponding points}\} \tag{4.13}$$

After that we use sum of squared distance of the corresponding point pairs as the error to be minimized. The minimization problem can be formulated as:

$$(\mathbf{R}^*, \mathbf{t}^*) = \underset{\mathbf{R},\mathbf{t}}{argmin} \sum_{(i,j)\in C} \|\mathbf{R}\mathbf{a}_i + \mathbf{t} - \mathbf{b}_j\|_2^2 \tag{4.14}$$

To solve this problem efficiently, we normally use the singular value decomposition (SVD) which guarantees the accuracy and speed during a large number of iterations. The ICP algorithm is summarized in the following algorithm 3:

Algorithm 3 Iterative Closest Point

Require: Source point cloud $A = \{\mathbf{a}_i\}_{i=1}^N$, target point cloud $B = \{\mathbf{b}_i\}_{i=1}^N$, maximum number of iterations k_{max}, threshold ϵ
Ensure: Rigid transformation $(\mathbf{R}, \mathbf{t})$

```
Initialize (R, t) with coarse pose estimation
for k = 1 to k_max do
    A* ⟵ Trans(A, (R, t))
    C = {(i, j) | a_i ∈ A and b_j ∈ B} ⟵ NearstSearch(A*, B)
    (R, t) ⟵ SVD(C)
    e = 1/|C| Σ_{(i,j)∈C} ||R a_i + t − b_j||_2^2
    if e < ε then
        Break
    end if
end for
```

Theoretically, the ICP algorithm can converge to the global optimal solution if the initial pose is close enough to the ground truth pose. However, in practice, the ICP algorithm is sensitive to the initial pose and can easily get stuck in the local optimal solution. Because of that , we use the coarse pose estimation from the diffusion model as the initial pose and run the ICP algorithm within an acceptable number of iterations to have a good tradeoff of inference speed and accuracy.

4.3. Experiments

In this section, we first introduce the datasets we use in our experiments and then we show the details of the training and evaluation of the pose hypotheses diffusion model and its feature extractors on the datasets. We compare our model with other methods and show the ablation study of our model.

4.3.1. Datasets

LINEMOD Dataset

The LINEMOD dataset [74] is a widely used dataset for the 6 DoF pose estimation. It contains 15 objects containing ape, bench vise, bowl, can, cat, driller, duck, glue, hole puncher, iron, lamp, phone, camera, and egg box. These objects are texture-less with discriminative color, shape and size and are placed in a cluttered scene. As the training dataset, it provide for each object 1312 RGB-D images from different view points annotated with the ground truth. These training data are based on the synthetic CAD Model without any occlusion and noise. There are also vividly rendered training data placed together in the scene to simulate the real scenario. The Benchmark for 6D Pose Estimation (BOP) [75]introduced the training data generated from an open-source, light-weight, procedural and photorealistic (PBR) renderer BlenderProc[76]into the LINEMOD dataset. This set of PBR-BlenderProc4BOP training images are sorted in 50 different scenes with 50000 images in total which are rendered with different lighting, occlusion. The test dataset are captured with the Kinect sensor. The following figures 4.9 show the training data and test data of the LINEMOD dataset.

Figure 4.9.: LIMEMOD dataset. Left: training data with Synthetic CAD Model; Middle: training data rendered with PBR-BlenderProc4BOP; Right: test data captured with Kinect Sensor

LINEMOD-O Dataset

To evaluate the model on the occluded scene, Brachmann et al. [16] refined the benchvise partion of the LINEMOD dataset with occluded objects and annotated ground truth pose under different light conditions to make the pose estimation more challenging. The meaning of the high-level occulusion data is to simulate the real scenario where the object is partially

occluded by other objects, so that the model can learn the robustness of the pose estimation and generalize better to the real world data. The following figure 4.10 shows the low-level occluded scene and high-level occluded scene.

Figure 4.10.: Comparison of LINEMOD-O dataset with LINEMOD dataset. Left: low-level occluded scene; Right: high-level occluded scene

Data Format

In order to unify the data format of the different dataset for 6 DoF pose estimation task. We follow the data format of the BOP dataset. The structure of the dataset is shown in the following table 4.1.

Table 4.1.: Dataset Structure of BOP Format

Directory	Contents
DATASET_NAME	
|— camera[_TYPE].json	Camera parameters
|— dataset_info.json	Dataset-specific information
|— test_targets_bop19.json	Test targets for the BOP Challenge
|— models[_MODELTYPE][_eval]	
| |— models_info.json	Dimension and symmetry
| |— obj_OBJ_ID.ply	3D file of the object
|— train|val|test[_TYPE]	Training/Validation/Test images
| |— SCENE_ID|OBJ_ID	
| | |— scene_camera.json	Camera parameters of the scene
| | |— scene_gt.json	Ground truth annotation
| | |— scene_gt_info.json	Meta information
| | |— depth	Depth images
| | |— mask	Masks of object silhouettes
| | |— mask_visib	Masks of the visible parts of object silhouettes
| | |— rgb|gray	Color/gray images

Clarification of the naming in table 4.1:

- models[_MODELTYPE]_eval: "Uniformly" resampled and decimated 3D object models used for calculation of errors of object pose estimates.

- MODELTYPE, TRAINTYPE, VALTYPE and TESTTYPE are optional and used if more data types are available (e.g. images from different sensors).

Camera parameters in scene_camera.json:

- cam_K - 3x3 intrinsic camera matrix K (saved row-wise).
- depth_scale - Multiply the depth image with this factor to get depth in mm.
- cam_R_w2c (optional) - 3x3 rotation matrix R_w2c (saved row-wise).
- cam_t_w2c (optional) - 3x1 translation vector t_w2c.
- view_level (optional) - Viewpoint subdivision level.

Ground truth annotation in scene_gt.json:

- obj_id - Object ID.
- cam_R_m2c - 3x3 rotation matrix R_m2c (saved row-wise).
- cam_t_m2c - 3x1 translation vector t_m2c.

Meta information of the ground truth in scene_gt_info.json:

- bbox_obj - 2D bounding box of the object silhouette given by (x, y, width, height), where (x, y) is the top-left corner of the bounding box.
- bbox_visib - 2D bounding box of the visible part of the object silhouette.
- px_count_all - Number of pixels in the object silhouette.
- px_count_valid - Number of pixels in the object silhouette with valid depth.
- px_count_visib - Number of pixels in the visible part of the object silhouette.
- isib_fract - The visible fraction of the object silhouette.

Data Augmentation

Although the quantity of the training data from the BlenderProc is large enough, we still need some data augmentation tricks to make the model robuster against the noise and invalid data. The data augmentation for point cloud in our case is firstly the random rotation of the partial point cloud and CAD model with the modification of the ground truth transformation. And we add noise to points with a Gaussian distribution scaled by a factor k_n related to the scale of the objects to simulate the real-world data captured by the depth sensor. And the symmetry of the object is also considered, that we generate extra data by flipping or rotating the symetrical CAD models with the corresponding ground truth.

To remove the noise and outliers in the point cloud. We use the statistical outlier filter in the data preprocessing stage. The statistical outlier filter is an algorithm that removes outliers from a point cloud based on the statistical analysis of point neighborhoods. The algorithm removes points that are further away from their neighbors compared to the average for the point cloud. The filter is controled by the the number of neighbors n_{nb} and the standard deviation multiplier σ. The effect of the algorithm is shown in figure.

After that the point cloud is then downsampled to 1000 points to reduce the computational cost but still maintain the geometry information of the object. The whole original training

dataset is divided into the training part (90%) and validation part (10%) randomly and together with the separate test dataset we get the dataset we preprocessed for the training and evaluation of our models.

As the 2D image is combined with 3D point cloud in the pose estimation task and we don't need to train the 2D feature extractor, we only use the data augmentation of the random rotaion during the training of the 3D feature extractor. Because the 2D image with 3D depth uniquely defines uniquely the 6 DoF pose of the object. So during the training of the pose estimation task we don't let orginal point cloud rotate, but still add noise and consider the symmetry of the object.

4.3.2. 2D Feature Extractor

Pretrained Model

DINO, as our 2D feature encoder, provides many pretrained weight of the backbone with different number of parameters and backbone architecture which are pretrained on ImageNet for general image downstream tasks. The evaluation of the self-supervised architecture is using the linear and k-NN classifier on the feature extracted from the model. The following table 4.2 shows the evaluation results of the different pretrained backbones on the ImageNet dataset according to the original paper [59].

Table 4.2.: Evaluation of the pretrained DINO model on ImageNet

Backbone	#Params.	Dim.	#Layers	#Heads	Patch Size	k-NN	Linear
ViT-S/16	21M	384	12	6	16	74.5	77.0
ViT-S/8	21M	384	12	6	8	78.3	79.7
ViT-B/16	85M	768	12	12	16	76.1	78.2
ViT-B/8	85M	768	12	12	8	77.4	80.1

After we register the pretrained 2D backbone to our model, we need to feed the image with some preprocessing into the ViT. The image is first cropped to a bounding box with the object we want to estimate its pose. Then we resize the cropped image to the size of 224x224 and normalize the image with the mean and standard deviation of the ImageNet dataset, while the ViT is pretrained on the ImageNet. The mean and standard deviation of the RGB channels are $(0.485, 0.456, 0.406)$ and $(0.229, 0.224, 0.225)$ respectively. After that, the image is processed following the ViT architecture and as output we get the feature we can use in the pose estimation task.

The 2D encoder need to be lightweight since it is just a downstream task for our diffusion model, so in our setup we don't use the backbone with a very deep network and large number of parameters, e.g.,ViT-L or ViT-G. And it needs also to be well generalized, because the image in our case is not seen during the training of the 2D encoder. ImageNet used in DINO contains more than 1.4 million annotated images from different categories, which has the capability to let the network learn the latent feature expression from the majority of the real-life objects.

Evaluation

Since we use the 2D feature not for the classification or segmentation tasks, but as the input for the pose estimation where the feature is the latent expression of pose that is hard to quantitatively evaluate, we evaluate the influence of the 2D feature extractor together with the pose estimation diffusion model later. Instead, the attention map of the input image can be visualized to indirectly evaluate the network and validate whether the Transformer can grasp the important part of the image. The attention map is the output of the last attention layer in the Transformer encoder. Figure 4.11 shows the attention maps of 3 objects as examples fed to the ViT.

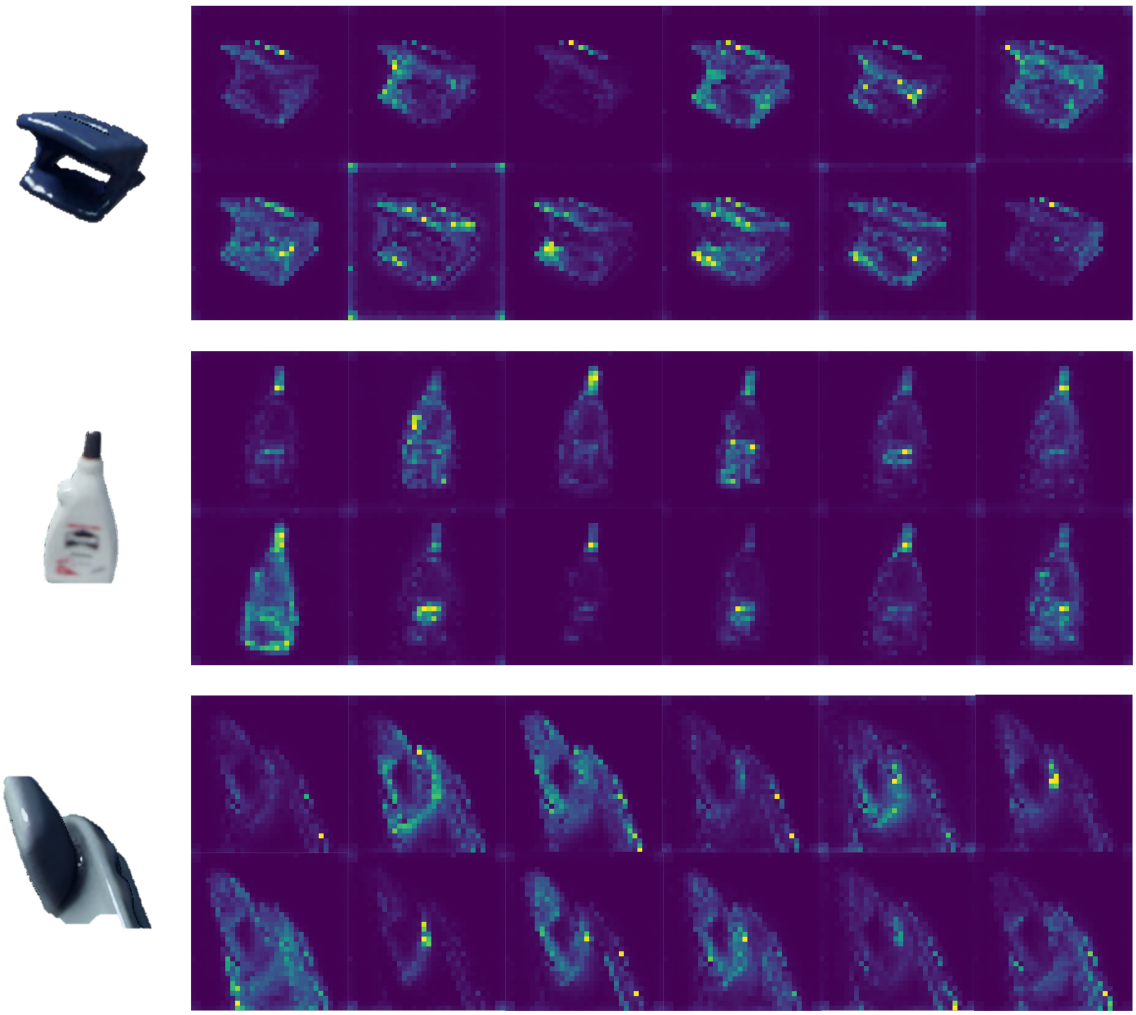

Figure 4.11.: Visualization of the attention maps of the input image, where 12 heatmaps represent 12 attension heads output in transformer. Top: hole puncher; Middle: glue; Bottem: phone (occluded)

By Comparing the outputs from different scale of ViT models and patch size, we can find that the patch size significantly determine the attention map. Because the object we dealing with has relative simple geometry and some times textureless, the transformer can not encode the image in a very accurate way for the pose difference if the patch size is too big. The following figure 4.12 shows the attention maps of the same image fed to the ViT with different patch

size and scale of ViTs.

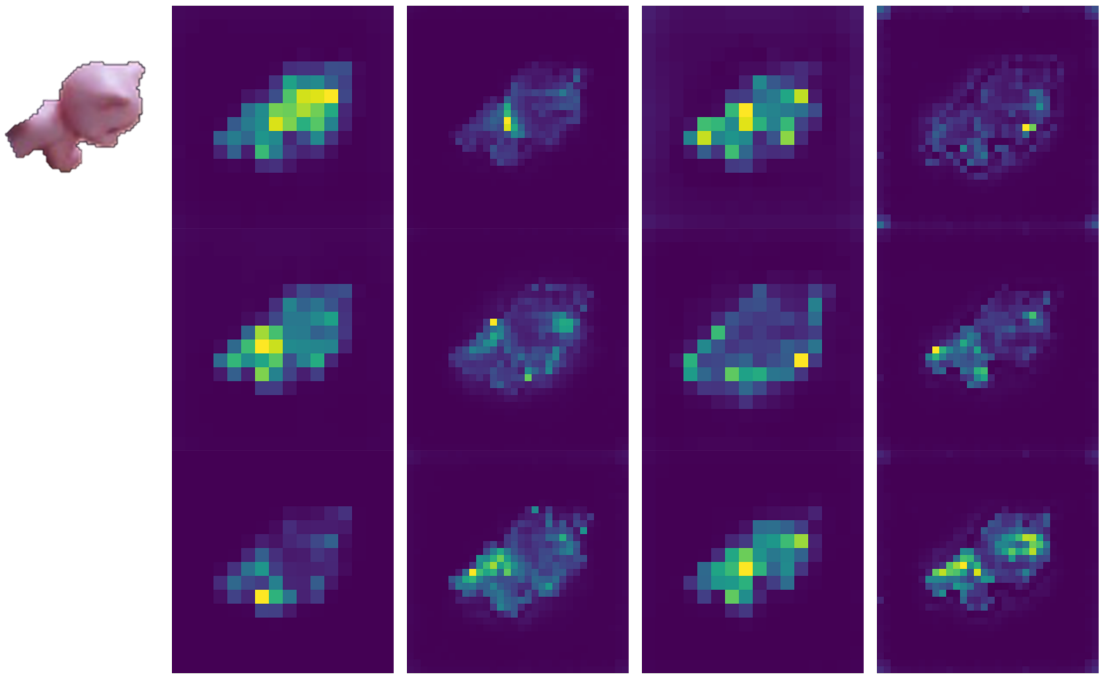

Figure 4.12.: Attension maps of the different scale of ViTs and patch size. From left to right: ViT-S/16, ViT-S/8, ViT-B/16, ViT-B/8. Each column represents 3 random attention heads.

From the figure 4.12 we can see that the key points of the cat e.g., ears, tail and eyes are not precisely localized if the patch size is not small enough. For the classification task is this drawback not that obvious, because the network can learn the latent feature from the majority of the image. But for the pose estimation task, the network need to learn the latent feature from the key points or edges of the object, which is more sensitive to the resolution.

Considering the tradeoff of the inference speed and performance, we use ViT-B/8 in our experiments which has a quite satistied performance and relative acceptable number of parameters. The ablation study of different setups will be shown later in this experiments section.

4.3.3. 3D Feature Extractor

Training

The 3D feature extractor is trained using a point cloud completion task. The reference partial point cloud is fed to the FoldingNet and the output is aligned with the CAD model transformed with the ground truth pose as the previous figure 4.8 shows. The loss function is the bi-directional Chamfer distance between the output point cloud and the transformed CAD model (see equation 4.12).

The model is trained on the single Nvidia Tesla V100-SXM2 video card with 16GB memory. We use the BlenderProc4BOP training dataset for LM which contains 50k images with 15 objects and we can further crop several objects from each image. To that end, we finally

have more than 560k seperate partial point clouds for training. After the tuning of the hyperparemeters, we use the batch size of 32 and the Adam optimizer with the cosine annealing learning rate scheduler. The initial learning rate is $1e^{-4}$ and the end-up learning rate is $1e^{-6}$. The model specific hyperparemeter is the number of the folding MLP layers, which is by default set to 2 in our case. The number of k-nearest neighbor is set to 16 and the number of the graph-based local maxpooling layers is set to 2. The reconstructed point number is set to 2025, whose square root is an integer responding to the size of the 2D grid. The dimension of the latent feature is set to 512 for the downstream task. The model is trained for 200k iteration which is about 114 epochs.

Evaluation

The evaluation metric of the point cloud completion task is the Chamfer distance (CD) between the output point cloud and the ground truth point cloud. The following graph shows the training, validation and test loss of the model during the whole training process. And the CD of each objects in the test dataset is shown in table

——————————image——————————-

——————————table——————————-

With the visualization of the reconstructed point cloud, we can valid that the model has the capability to localize the partial point cloud and complete the missing part of the object. The implicit expression of the 6 DoF pose as well as the object class is learned by the model without giving the class label, so that can use the feature to represent the pose and object class of the reference input in the diffusion model.

——————————image——————————-

The following figure shows the development of the output from FoldingNet decoder during the training process.

——————————image——————————-

From the visualization we can find that the shape of the reconstructed point cloud is getting more and more similar to the ground truth, but the difference exists because of the capability of the backbone on data with heavy occlusion and noise. Figure shows the result of the point cloud completion task on the overfitting case compared with the generalized version on the whole dataset, which has a quite satisfied result. The visualization of the overfitting case on one objects is illustrated in figure.

——————————image——————————-

——————————image——————————-

4.3.4. Training

The training of the diffusion network for the pose hypotheses estimation can be done after the 2D and 3D feature encoders are pretrained (for 2D feature extractor is the checkpoint of the pretrained model loaded from the public cloud storage). We load the checkpoint of the encoders and freeze the network of these two parts to avoid the parameters of pretrained model being updated during the training of the diffusion network.

The training is also running on the single Nvidia Tesla V100-SXM2 16G video card. Like setup of the 3D encoder, we use the BlenderProc4BOP training dataset and divide the original data into 90%/10% partitions for training and validation. After tuning the hyperparemeters from the experiments, we use the following setup (table 4.3) to represent the pose hypotheses diffusion model.

Table 4.3.: Hyperparemeters of the pose hypotheses diffusion model

Training param.	Conf.	Diffusion	Conf.	Backbone	Conf.
Batch size	16	Steps	400	Model	Transformer
Optimizer	Adam	β_1	$1e^{-4}$	#Layers	8
Scheduler	CA	β_T	$2e^{-2}$	#Heads	4
Initial lr.	$1e^{-4}$	Scheduler	Linear	Hidden dim.	512
End-up lr.	$1e^{-6}$	Σ_θ	Learnable	FF dim.	2048
#Iterations	200k	Rotaion	6D	TE dim.	256
		Loss func.	MSE	PE dim.	512
				2D dim.	768
				3D dim.	512
				Feat. fusion	Add + adaLN

Clarification of some hyperparemeters in table 4.3:

- CA - Cosine annealing learning rate scheduler.
- β_1 - Initial value of the variance schedule at time step 1.
- β_T - Final value of the variance schedule at time step T.
- Scheduler (Diffusion) - Variance scheduler of the forward diffusion process which represents the change of β with the time step t.
- Σ_θ - Diagonal variance of the reverse process which is fixed in the orignial paper [30] but can be optimized to be learnable [34].
- Rotation - Representation of the rotation matrix for 6 DoF pose. We use the 6D continuous representation in our case. Recall the section 2.1.2.
- #layers - Number of the multi-head attention layers.
- #heads - Number of the attention heads.
- Hidden dim. - Dimension of the hidden layer in the Transformer encoder.
- FF dim. - Dimension of the feed forward layers in the Transformer encoder, which is chosen to be 4 times of the hidden dim.
- TE dim. - Dimension of the time embedding in the diffusion model.
- PE dim. - Dimension of the postion embedding of the input tokens for Transformer encoder.
- 2D dim. - Dimension of the 2D feature extracted from the pretrained 2D encoder.
- 3D dim. - Dimension of the 3D feature extracted from the pretrained 3D encoder.

- Feat. fusion - The operation of the feature fused into the backbone. In our case, we first use the addition of the 2D, 3D features as well as the time embedding and then utilize the adaptive layer normalization (AdaLN) to integrate the features into the network. Recall the section 4.2.2.

The loss function is defined by the mean square error (MSE) between the noise ϵ added at a particular time step t and the estimated noise ϵ_θ. The loss function is formulated as:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{B} \sum_{t \in \mathfrak{T}_B} \|\epsilon_t - \epsilon_\theta(\mathbf{x}_0, \epsilon, t, y)\|^2 \quad , \mathfrak{T}_B \subseteq \mathfrak{T} = \{1, \ldots, T\} \tag{4.15}$$

where B is the batch size, $\mathfrak{T}_B$ is the subset of the whole time steps with the batch size and ϵ_θ is the estimated noise depending on the input $\mathbf{x}_0$, the noise ϵ, the time step t and the condition y (see section 2.2.2). The loss is calculated with the MSE of the randomly sampled subset controled by the batch size instead of the whole time steps, but it will cover each time step during the training process. Figure shows the loss of the model during the training process.

————————image————————-

4.3.5. Evaluation

Evaluation Metrics

No matter we use which form of the rotation representation during the training phase that is introduced in section 2.1.2, we transform it back to the rotation matrix $\mathbf{R}$ to evaluate the pose estimation together with the tranlation, in order to meet the requirement of the data format of BOP. The pose is represented by a 4x4 matrix $\mathbf{T} = [\mathbf{R}, \mathbf{t}, \mathbf{0}, 1]$, where $\mathbf{R}$ is a 3x3 rotation matrix and $\mathbf{t}$ is a 3x1 translation vector. As evaluation metrics, we adopt the same metrics used in BOP Challenge 2019/2020/2022/2023 [75, 77]. The following metrics are used to evaluate the pose estimation. Visible Surface Discrepancy (VSD):

$$e_{\text{VSD}}\left(\hat{D}, \bar{D}, \hat{V}, \bar{V}, \tau\right) = \text{avg}_{p \in \hat{V} \cup \bar{V}} \begin{cases} 0 & \text{if } p \in \hat{V} \cap \bar{V} \wedge \left|\bar{D}(p) - \hat{D}(p)\right| < \tau \\ 1 & \text{otherwise} \end{cases} \tag{4.16}$$

An object model is rendered in two poses: the estimated pose $\hat{\mathbf{P}}$ and the ground truth pose $\bar{\mathbf{P}}$. And the result of the rendering is two distance maps $\hat{S}$ and $\bar{S}$. The distance maps are compared with the distance map S_I of the test image I to calculate the visibility masks $\hat{V}$ and $\bar{V}$. The error is control by a misalignment tolerence τ.

Maximum Symmetry-Aware Surface Distance (MSSD):

$$e_{\text{MSSD}}\left(\hat{\mathbf{P}}, \bar{\mathbf{P}}, S_M, V_M\right) = \min_{\mathbf{S} \in S_M} \max_{\mathbf{x} \in V_M} \left\|\hat{\mathbf{P}}\mathbf{x} - \bar{\mathbf{P}}\mathbf{S}\mathbf{x}\right\|_2 \tag{4.17}$$

where S_M is the set of all symmetry transformations of the object model and V_M is the set of model vertices. The MSSD is the minimum of the maximum symmetric-aware surface distance between the estimated pose $\hat{\mathbf{P}}$ and the ground truth pose $\bar{\mathbf{P}}$. The maximum distance between the model vertices is important for robotic application, where the maximum surface deviation strongly indicates the chance of a successful grasp.

Maximum Symmetry-Aware Projection Distance (MSPD):

$$e_{\text{MSPD}}\left(\hat{\mathbf{P}}, \bar{\mathbf{P}}, S_M, V_M\right) = \min_{\mathbf{S}\in S_M} \max_{\mathbf{x}\in V_M} \left\| \text{proj}\left(\hat{\mathbf{P}}\mathbf{x}\right) - \text{proj}\left(\bar{\mathbf{P}}\mathbf{S}\mathbf{x}\right) \right\|_2 \tag{4.18}$$

The proj() is the 2D projection which results in pixels and other variables are identical to the MSSD. The MSPD considers global object symmetries and replaces the average by the maximum distance to increase robustness against the sampling of the object model compared with the previous method [78].

After the error functions are determined, we use the threshold of the error to define the correctness of the pose estimation. The pose is correctly estimated if the error $e < \theta_e$, where $e \in \{e_{\text{VSD}}, e_{\text{MSSD}}, e_{\text{MSPD}}\}$ and θ_e is the threshold of correctness.

AR_{VSD} is the average recall rates calculated for the misalignment tolerence τ from 5% to 50% of the object diameter with a step of 5%, and the threshold of correctness θ_{VSD} ranging from 0.05 to 0.5 with a step of 0.05. AR_{MSSD} is the average recall rate calculated for the threshold of correctness θ_{MSSD} ranging from 0.05 to 0.5 of the object diameter with a step of 0.05. AR_{MSPD} is the average recall rate calculated for the threshold of correctness θ_{MSPD} ranging from $5r$ to $50r$ with a step of $5r$, where $r = w/640$ and w is the image width in pixels. The average recall rate is defined as:

$$\text{AR} = \left(\text{AR}_{\text{VSD}} + \text{AR}_{\text{MSSD}} + \text{AR}_{\text{MSPD}}\right)/3 \tag{4.19}$$

————————————why not add————————————-

Baseline

Pose Estimation Results

4.3.6. Ablation Study

5. Correspondance Diffusion

5.1. Introduction

5.2. Methodology

5.2.1. Pipeline

5.2.2. Models

5.3. Experiments

5.3.1. Datasets

5.3.2. Evaluation

Evaluation Metrics

6. Discussion

7. Conclusion

A. Additionally

You may do an appendix

List of Figures

List of Tables

Bibliography

[1] C. Jones, A. Smith and E. Roberts, "Article title," in *Proceedings Title*, vol. II. IEEE, 2003, pp. 803–806.

[2] S. Peng, Y. Liu, Q. Huang, X. Zhou and H. Bao, "PVNet: Pixel-Wise Voting Network for 6DoF Pose Estimation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA: IEEE, Jun. 2019, pp. 4556–4565. [Online]. Available: https://ieeexplore.ieee.org/document/8954204/

[3] F. Manhardt, "Towards monocular 6d object pose estimation," Ph.D. dissertation, Technische Universität München, 2021.

[4] Y. Xiang, T. Schmidt, V. Narayanan and D. Fox, "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes," *CoRR*, vol. abs/1711.00199, 2017. [Online]. Available: http://arxiv.org/abs/1711.00199

[5] H. A. Hashim, "Special orthogonal group so(3), euler angles, angle-axis, rodriguez vector and unit-quaternion: Overview, mapping and challenges," *ArXiv preprint ArXiv:1909.06669*, 2019.

[6] V. Mansur, S. Reddy, S. R and R. Sujatha, "Deploying complementary filter to avert gimbal lock in drones using quaternion angles," in *2020 IEEE International Conference on Computing, Power and Communication Technologies (GUCON)*, 2020, pp. 751–756.

[7] G. Terzakis, M. Lourakis and D. Ait-Boudaoud, "Modified rodrigues parameters: An efficient representation of orientation in 3d vision and graphics," *Journal of Mathematical Imaging and Vision*, vol. 60, 03 2018.

[8] Y. Zhou, C. Barnes, L. Jingwan, Y. Jimei and L. Hao, "On the continuity of rotation representations in neural networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[9] Y. Zhu, M. Li, W. Yao and C. Chen, "A review of 6d object pose estimation," in *2022 IEEE 10th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, vol. 10, 2022, pp. 1647–1655.

[10] H. Cao, L. Dirnberger, D. Bernardini, C. Piazza and M. Caccamo, "6impose: Bridging the reality gap in 6d pose estimation for robotic grasping," 2023.

[11] Z. Qin, H. Yu, C. Wang, Y. Guo, Y. Peng and K. Xu, "Geometric transformer for fast and robust point cloud registration," 2022.

[12] K. Fu, S. Liu, X. Luo and M. Wang, "Robust point cloud registration framework based on deep graph matching," 2021.

[13] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, pp. 239–256, 1992. [Online]. Available: https://api.semanticscholar.org/CorpusID:21874346

[14] R. A. Rill and K. Faragó, "Collision avoidance using deep learning-based monocular vision," *SN Computer Science*, vol. 2, 09 2021.

[15] G. Marullo, L. Tanzi, P. Piazzolla and E. Vezzetti, "6d object position estimation from 2d images: a literature review," *Multimedia Tools and Applications*, vol. 82, pp. 1–39, 11 2022.

[16] E. Brachmann, "6D Object Pose Estimation using 3D Object Coordinates [Data]," 2020. [Online]. Available: https://doi.org/10.11588/data/V4MUMX

[17] T. Hodaň, P. Haluza, Š. Obdržálek, J. Matas, M. Lourakis and X. Zabulis, "T-LESS: An RGB-D dataset for 6D pose estimation of texture-less objects," *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017.

[18] A. Kendall, M. Grimes and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," 2016.

[19] G. Pavlakos, X. Zhou, A. Chan, K. G. Derpanis and K. Daniilidis, "6-dof object pose from semantic keypoints," 2017.

[20] T. Hodan, V. Vineet, R. Gal, E. Shalev, J. Hanzelka, T. Connell, P. Urbina, S. N. Sinha and B. Guenter, "Photorealistic image synthesis for object instance detection," 2019.

[21] A. Lamb, "A brief introduction to generative models," 2021.

[22] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative adversarial networks," 2014.

[23] A. Borji, "Pros and cons of gan evaluation measures," 2018.

[24] M. Arjovsky, S. Chintala and L. Bottou, "Wasserstein gan," 2017.

[25] T. Miyato, T. Kataoka, M. Koyama and Y. Yoshida, "Spectral normalization for generative adversarial networks," 2018.

[26] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2022.

[27] D. P. Kingma, T. Salimans and M. Welling, "Variational dropout and the local reparameterization trick," 2015.

[28] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," 2016.

[29] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," 2015.

[30] J. Ho, A. Jain and P. Abbeel, "Denoising diffusion probabilistic models," 2020.

[31] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever and M. Chen, "Glide: Towards photorealistic image generation and editing with text-guided diffusion models," 2022.

[32] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu and M. Chen, "Hierarchical text-conditional image generation with clip latents," 2022.

[33] L. Weng, "What are diffusion models?" *lilianweng.github.io*, Jul 2021. [Online]. Available: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

[34] A. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," 2021.

[35] Y. Song and S. Ermon, "Generative modeling by estimating gradients of the data distribution," 2020.

[36] J. Ho and T. Salimans, "Classifier-free diffusion guidance," 2022.

[37] L. Yang, Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, W. Zhang, B. Cui and M.-H. Yang, "Diffusion models: A comprehensive survey of methods and applications," 2023.

[38] C. Saharia, J. Ho, W. Chan, T. Salimans, D. J. Fleet and M. Norouzi, "Image super-resolution via iterative refinement," *arXiv:2104.07636*, 2021.

[39] J. Ho, C. Saharia, W. Chan, D. J. Fleet, M. Norouzi and T. Salimans, "Cascaded diffusion models for high fidelity image generation," *arXiv preprint arXiv:2106.15282*, 2021.

[40] S. Gao, X. Liu, B. Zeng, S. Xu, Y. Li, X. Luo, J. Liu, X. Zhen and B. Zhang, "Implicit diffusion models for continuous super-resolution," 2023.

[41] A. Lugmayr, M. Danelljan, A. Romero, F. Yu, R. Timofte and L. V. Gool, "Repaint: Inpainting using denoising diffusion probabilistic models," 2022.

[42] C. Saharia, W. Chan, H. Chang, C. A. Lee, J. Ho, T. Salimans, D. J. Fleet and M. Norouzi, "Palette: Image-to-image diffusion models," 2022.

[43] G. Kwon and J. C. Ye, "Diffusion-based image translation using disentangled style and content representation," 2023.

[44] S. Luo and W. Hu, "Diffusion probabilistic models for 3d point cloud generation," 2021.

[45] X. Zeng, A. Vahdat, F. Williams, Z. Gojcic, O. Litany, S. Fidler and K. Kreis, "Lion: Latent point diffusion models for 3d shape generation," 2022.

[46] Z. Lyu, Z. Kong, X. Xu, L. Pan and D. Lin, "A conditional point diffusion-refinement paradigm for 3d point cloud completion," 2022.

[47] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[48] A. Radford and K. Narasimhan, "Improving language understanding by generative pre-training," 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:49313245

[49] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave and G. Lample, "Llama: Open and efficient foundation language models," 2023.

[50] H. Zou, Z. M. Kim and D. Kang, "A survey of diffusion models in natural language processing," 2023.

[51] J. Austin, D. D. Johnson, J. Ho, D. Tarlow and R. van den Berg, "Structured denoising diffusion models in discrete state-spaces," 2023.

[52] E. Hoogeboom, D. Nielsen, P. Jaini, P. Forré and M. Welling, "Argmax flows and multinomial diffusion: Learning categorical distributions," 2021.

[53] X. L. Li, J. Thickstun, I. Gulrajani, P. Liang and T. B. Hashimoto, "Diffusion-lm improves controllable text generation," 2022.

[54] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger and I. Sutskever, "Learning transferable visual models from natural language supervision," 2021.

[55] R. Rombach, A. Blattmann, D. Lorenz, P. Esser and B. Ommer, "High-resolution image synthesis with latent diffusion models," 2022.

[56] L. Zhang, A. Rao and M. Agrawala, "Adding conditional control to text-to-image diffusion models," 2023.

[57] B. Poole, A. Jain, J. T. Barron and B. Mildenhall, "Dreamfusion: Text-to-3d using 2d diffusion," 2022.

[58] O. Ronneberger, P. Fischer and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[59] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski and A. Joulin, "Emerging properties in self-supervised vision transformers," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.

[60] Y. Yang, C. Feng, Y. Shen and D. Tian, "Foldingnet: Point cloud auto-encoder via deep grid deformation," 2018.

[61] P. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[62] J. Song, C. Meng and S. Ermon, "Denoising diffusion implicit models," 2022.

[63] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, "Attention is all you need," 2023.

[64] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021.

[65] E. Perez, F. Strub, H. de Vries, V. Dumoulin and A. Courville, "Film: Visual reasoning with a general conditioning layer," 2017.

[66] W. Peebles and S. Xie, "Scalable diffusion models with transformers," *arXiv preprint arXiv:2212.09748*, 2022.

[67] A. Kazemnejad, "Transformer architecture: The positional encoding," *kazemnejad.com*, 2019. [Online]. Available: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

[68] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[69] C. R. Qi, H. Su, K. Mo and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," 2017.

[70] C. R. Qi, L. Yi, H. Su and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," 2017.

[71] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette and L. J. Guibas, "Kpconv: Flexible and deformable convolution for point clouds," 2019.

[72] Y. Shen, C. Feng, Y. Yang and D. Tian, "Mining point cloud local structures by kernel correlation and graph pooling," 2018.

[73] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,"

Commun. ACM, vol. 24, no. 6, p. 381–395, jun 1981. [Online]. Available: https://doi.org/10.1145/358669.358692

[74] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige and N. Navab, "Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes," vol. 7724, 10 2012.

[75] T. Hodan, F. Michel, E. Brachmann, W. Kehl, A. G. Buch, D. Kraft, B. Drost, J. Vidal, S. Ihrke, X. Zabulis, C. Sahin, F. Manhardt, F. Tombari, T.-K. Kim, J. Matas and C. Rother, "Bop: Benchmark for 6d object pose estimation," 2018.

[76] M. Denninger, M. Sundermeyer, D. Winkelbauer, Y. Zidan, D. Olefir, M. Elbadrawy, A. Lodhi and H. Katam, "Blenderproc," 2019.

[77] T. Hodan, M. Sundermeyer, B. Drost, Y. Labbe, E. Brachmann, F. Michel, C. Rother and J. Matas, "Bop challenge 2020 on 6d object localization," 2020.

[78] E. Brachmann, F. Michel, A. Krull, M. Y. Yang, S. Gumhold and C. Rother, "Uncertainty-driven 6d pose estimation of objects and scenes from a single rgb image," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 3364–3372.

Declaration

Herewith, I declare that I have developed and written the enclosed thesis entirely by myself and that I have not used sources or means except those declared.

This thesis has not been submitted to any other authority to achieve an academic grading and has not been published elsewhere.

Stuttgart, TBD Date of sign. Student's name TBD