Advanced-Level Training Problem Set for Computer Vision, Graphics, and Robotics

Part I – Foundation, 2D Vision, 3D Vision

Su Lab @ UCSD

Contents

Chapter 1

Linear Algebra and Convex Optimization

1.1 SVD, KKT, QCQP

We consider the following constrained least square problem

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|Ax-b\|_2^2 \\ \text{subject to} \quad & x^T x \le \epsilon, \end{aligned}$$

where $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m, \epsilon \in \mathbb{R}, \epsilon > 0$. The variable is $x \in \mathbb{R}^n$.
The solution to this problem will be used as a subroutine in subsequent geometry homeworks and we need an efficient solver. The solution to this optimization problem has to satisfy the Karush–Kuhn–Tucker (KKT) conditions:

$$\begin{cases} \nabla_x \mathscr{L}(x,\lambda) = 0 & (1) \\ x^T x \le \epsilon & (2) \\ \lambda \ge 0 & (3) \\ \lambda(x^T x - \epsilon) = 0 & (4) \end{cases}$$

where $\mathscr{L}(x,\lambda) = \frac{1}{2}\|Ax-b\|_2^2 + \lambda(x^T x - \epsilon)$ is the Lagrangian of the problem.

1. Write down the gradient of the Lagrangian $\nabla_x \mathscr{L}$.

   Next, We will solve this problem by considering 2 cases based on Condition (2).

2. Case 1: $x^T x < \epsilon$. Then, $\lambda = 0$ by Condition (4). To satisfy the KKT condition (1), it is equivalent to solving the unconstrained least square problem with objective $\frac{1}{2}\|Ax-b\|_2^2$. If its solution satisfies (2), we are done. Write down the closed-form solution to the unconstrained least-square problem. You do not need to show intermediate steps. (1pt)

3. Case 2: $x^T x = \epsilon$.

(a) Set $\nabla_x \mathscr{L} = 0$, express x in terms of λ, i.e. $x = h(\lambda)$ (1pt)

(b) Prove $h(\lambda)^T h(\lambda)$ is monotonically decreasing for $\lambda \geq 0$. *Hint:* You might need the fact that $A^T A = U \Lambda U^T$. (2pt)

By the monotone property of $h(\lambda)^T h(\lambda)$, we can solve $x^T x = \epsilon$ by line search over λ (e.g., bisection method or Newton's iterative method). You will implement it in the following programming assignment.

4. (Programming assignment) Solve a provided instance of this problem.

   - You can load the data by

     ```
     npz = np.load('./HW0_P1.npz')
     A = npz['A']
     b = npz['b']
     eps = npz['eps']
     ```

   - You are **not** allowed to use external optimization libraries that solve this problem directly. Linear algebra functions in numpy are allowed, e.g. numpy.linalg.eig, numpy.linalg.svd, numpy.linalg.lstsq, etc.

1.2 Probability

Given a triangle $\triangle ABC$, one common geometric processing question is how to sample n points uniformly inside the triangle.
A straight-forward idea is to interpolate the vertices by barycentric coordinates: we first sample $\alpha, \beta, \gamma \sim U([0,1])$, where $U([0,1])$ is the uniform distribution on $[0,1]$; then, we normalize them to obtain $\alpha' = \frac{\alpha}{\alpha+\beta+\gamma}$, $\beta' = \frac{\beta}{\alpha+\beta+\gamma}$, $\gamma' = \frac{\gamma}{\alpha+\beta+\gamma}$ so that $\alpha'+\beta'+\gamma' = 1$; finally, we obtain a sample $P = \alpha' A + \beta' B + \gamma' C$ inside $\triangle ABC$. However, this straight-forward idea is wrong: it does not assure that P is uniformly sampled in $\triangle ABC$.

1. We seek to rigorously disprove the above algorithm for a lower-dimensional setup, which samples points on a line segment $\overline{AB}$: If we sample $\alpha, \beta \sim U([0,1])$ and normalize by $\alpha' = \frac{\alpha}{\alpha+\beta}$ and $\beta' = \frac{\beta}{\alpha+\beta}$ to obtain $P = \alpha' A + \beta' B$, then P is not uniformly distributed between A and B. For simplicity, we assume that $A = 0$ and $B = 1$.

   - Show the cumulative density function of P, i.e., $\Pr(P \leq t)$. (Hint: $\Pr(A) = \int_x 1_{[x \in A]} f(x) \mathrm{d}x$ where $1_{[x \in A]}$ is the indicator function and $f(x)$ is the density function of random variable x.) (2pt)
   - Compute the value of the density function of P at $P = 0$ and $P = 0.5$. (1pt)

2. A correct algorithm for uniformly sampling points in $\triangle ABC$ is the following:

   (a) Sample $\alpha \sim U([0,1])$ and $\beta \sim U([0,1])$

   (b) $P' = A + \alpha(B - A) + \beta(C - A)$

   (c) If P' inside $\triangle ABC$, accept by letting $P = P'$. Otherwise, let $P = B + C - P'$.

   Question:

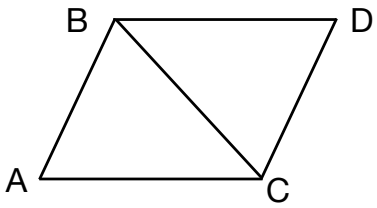

Figure 1.1: Figure for P2

- Prove that P' is uniformly distributed inside the parallelogram $ABDC$. (Hint: Show the density function pdf directly. Do not compute the cdf.) Hint 2: Suppose $\mathbf{y} = H(\mathbf{x})$ and H is bijective and differentiable, J is the Jacobian of $\mathbf{y}$ with respective to $\mathbf{x}$, $\mathbf{x}$ has density f, and $\mathbf{y}$ has density g, then $g(\mathbf{y}) = f(H^{-1}(\mathbf{y}))|\det(J^{-1})|$.) (2pt)
- Prove that P is uniformly distributed in $\triangle ABC$. (1pt)

3. Given a triangle whose vertices are at $A = (0,0)$, $B = (0,1)$, and $C = (1,0)$, write a program to simulate the wrong algorithm at the beginning of this problem and the correct algorithm shown above. Make a comparison of them by sampling 1000 points inside the triangle. Include the code and the plot in your submission. (2pt)

   For your convenience, we provide the following codes:

```
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon

pts = np.array([[0,0], [0,1], [1,0]])
def draw_background(index):
    # DRAW THE TRIANGLE AS BACKGROUND
    p = Polygon(pts, closed=True, facecolor=(1,1,1,0), edgecolor=(0, 0, 0))

    plt.subplot(1, 2, index + 1)

    ax = plt.gca()
    ax.set_aspect('equal')
    ax.add_patch(p)
    ax.set_xlim(-0.1,1.1)
    ax.set_ylim(-0.1,1.1)

# YOUR CODE HERE

draw_background(0)
# REPLACE THE FOLLOWING LINE USING YOUR DATA (incorrect method)
plt.scatter(0.4+0.2*np.random.randn(1000),
            0.4+0.2*np.random.randn(1000), s=3)

draw_background(1)
# REPLACE THE FOLLOWING LINE USING YOUR DATA (correct method)
```

```
plt.scatter(0.4+0.2*np.random.randn(1000),
            0.4+0.2*np.random.randn(1000), s=3)

plt.show()
```

Chapter 2

2D Deep Learning

For each problem, we provide dataset, and we give link to relevant papers.
Remember that, for pytorch models, always use nn.Modulelist instead of python lists to contain a list of modules.

2.1 Image Classification, Network Architectures, and Data Augmentation

Image classification is one of the most fundamental computer vision tasks. In this problem, we aim to explore the influence of architecture, data augmentation, and the amount of training data on model performance. **Please put all code to this problem in a single jupyter notebook.**
For the following tasks, use float32 for training, even though the tutorials might say use float16. *It is not advised to use float16 generally as it could often lead to unexpected numerical gradient issues for many other datasets and many other tasks (especially for Reinforcement Learning tasks, you should never use float16). Thus if you want to use float16, only do it after float32 results look reasonable.*
Paper reference: ResNet (`https://arxiv.org/pdf/1512.03385`), Transformer (`https://arxiv.org/pdf/1706.03762.pdf`), vision Transformer (ViT) (`https://arxiv.org/pdf/2010.11929`)

- First, implement **standard ResNet 18** (reference implementation) and train on **CIFAR-100**. Use a recently-proposed high-performance dataloader FFCV (tutorial: link1, link2) for dataset loading and processing. Use a batch size of 128 and train for 120 epochs with the following learning rate (lr) schedule: first 60 epochs, use lr=0.1; next 30 epochs, use lr=0.01; final 30 epochs, use lr=0.001. Use SGD optimizer and weight decay 1e-4 (CIFAR, ImageNet classification are the few places we use SGD for training ResNet; for other cases we use Adam). Use the following standard data augmentation for CIFAR:
  - `(RandomCrop(32, padding=4), RandomHorizontalFlip())` in `torchvision.transforms`
  - or you can use the FFCV (ffcv.transforms) equivalent `(RandomTranslate(padding=4), RandomHorizontalFlip())`

Report training and test accuracy of 3 different models, trained on (1) 5% of training data; (2) 20% of training data, and (3) all of training data, respectively. You should get >71% success rate when trained on the entire training data.

(As a side note, above data augmentation is different from ImageNet training because CIFAR image size is 32x32 while ImageNet image size is 256x256. In ImageNet, the standard way is to take a 224x224 crop of the 256x256 image and apply random horizontal flip)

(As a side note, standard ResNet for image classification uses Batch Normalization. However for other use cases, Batch Normalization should not always be used. For example, in image-based reinforcement learning, Batch Normalization should never be used (since batch statistics for RL varies significantly). In these cases Layer Normalization is a better choice.)

- Next, implement **Vision Transformer (ViT)** and train on CIFAR-100. For this part you only need to train on the entire training data. A reference ViT implementation is here. Make sure that you completely understand multi-head self-attention. Make sure that you can write code using tools like `einops` and operations like torch.einsum, which makes Transformer code readable.

  Use a 6 layer ViT with hidden dimension 512 and MLP dimension 1024. Use a batch size of 128 and standard data augmentation. Use a dropout of 0.1 and embedding dropout of 0.1 (empirically dropout significantly reduces overfitting in Transformers; if you do not know Dropout, Batch Normalization, Layer Normalization, etc yet, make sure that you understand them). Note that Transformer implementation uses Layer Normalization instead of Batch Normalization like in standard ResNet, because empirically for Transformers, the batch statistics vary significantly between batches, which leads to instability.

  Use Adam Optimizer for ViT. You need to tune the learning rate (in small batch-size Transformer training, the lr should be <1e-3, and sth around 3e-4, 1e-4, 5e-5 should work well). Use a linear learning rate decay for training. You also need to tune the patch size (since CIFAR input image size is 32x32) and the number of attention heads (this should be >=8 for a 6-layer Transformer with hidden dim 1024, but should not be too many). Compare the performance of different hyperparameter choices.

  (As a side note, when Google, Facebook etc train Transformers for NLP, they often use a very large batch size and therefore a higher learning rate (often at a scale of 0.01 in the initial stages of training after warmup). However, we don't have much compute compared to industry-level infrastructures, so our batch size should be smaller accordingly)

- Finally, investigate the effect of different data augmentations for both ResNet and ViT. Investigate `ColorJitter` in `torch.transforms` along with one of `GrayScale` or `GaussianBlur` (you need to briefly tune the hyperparameters for these data augmentations). For each data augmentation, compose on top of standard data augmentations in CIFAR, and report training and test performance (also state the hyperparameters used for these data augmentations). A document for these augmentations can be found at this link.

  (The reason for choosing to study these augmentations is that they are used in self-supervised learning approaches like MoCo and BYOL, with implementations in e.g. this link)

2.2 Instance Segmentation

Overview Instance segmentation is a basic and classical computer vision task of detecting instances and predicting a segmentation mask per instance. Object detection is an extension to image classification. From this perspective, instance segmentation can be regarded as a combination of object detection and semantic segmentation (within each instance). Thus, understanding instance segmentation requires knowledge of object detection and semantic segmentation.
The key challenge of instance segmentation (or object detection), different from image classification, is how to predict a varying number of instances. The common practice is to define a set of candidates, and solve classification (foreground/background) and regression problems (size, location, semantic class) for each candidate. The most classical learning-based approach is Mask R-CNN, which extends the classical object detector R-CNN to predict an instance mask for each object. It follows R-CNN to explicitly generate candidates (proposals) in a sliding window fashion. However, the implementation of proposal generation and label assignment is complicated, and there are many relevant hyper-parameters. Besides, it tends to output duplicate detection objects, and requires some post-processing methods, like non-maximum suppression (NMS). In the recent years, DETR implicitly generates candidates and uses a set loss between ground-truth and outputs. It simplifies the implementation, but may converge more slowly. There are several following-up works to accelerate the training of DETR. DETR can also be extended for instance segmentation.
The most widely used dataset is COCO. PASCAL VOC, a smaller dataset, is used in many earlier works. The most common evaluation metric is mean average precision (mAP). Notice that it is also the metric for object detection. "The evaluation metrics for detection with bounding boxes and segmentation masks are identical in all respects except for the IoU computation (which is performed over boxes or masks, respectively)."
We recommend two popular libraries: Detectron2 and MMDetection. Besides, torchvision also provides a simple Mask R-CNN implementation.

Problem In this problem, you are asked to train an instance segmentation network (Mask R-CNN or DETR) on two datasets, Linemod and YCB-Video. The datasets can be downloaded from BOP. For Linemod, please use PBR rendered images as the training set, and all test images as the test set. For YCB-Video, please use the original real training images as the training set, and all test images as the test set.
You need to submit a report to include:

- Hyper-parameters that matter according to your experiments
- Evaluation results (mAP) on the test sets
- Visualization of success and failure cases

In fact, these two datasets only contain one instance per category in each image. However, they still can serve as an instance segmentation dataset.

2.3 Variational Auto-Encoder (VAE)

- Finish problem “VAE” in `vae_gan_release/VAE_GAN.ipynb`, where you will implement a standard VAE (without any label conditioning) on SVHN dataset.

- Derive the conditional VAE (cVAE) objective, where we want to reconstruct the target **y** given source **x** (e.g. given class label, construct images in the class).

2.4 WGAN-GP with SNGAN-like Architecture

Source: UC Berkeley Deep Unsupervised Learning Spring 2020
Paper: https://arxiv.org/pdf/1704.00028.pdf, https://arxiv.org/pdf/1802.05957.pdf
Finish problem "WGAN-GP with SNGAN-like Architecture" in `vae_gan_release/VAE_GAN.ipynb`, which implements WGAN-GP on CIFAR-10 using architecture similar to SN-GAN. Pay attention to the hyperparameter used for training and the fact that discriminator is optimized 5x more frequently than generator (this is true for image generation; however the schedule is always dependent on the dataset and task being used; for other tasks like GAIL in reinforcement learning, discriminator should be updated much less frequently or otherwise the discriminator reward for policy update will be near zero)
(note: It's important to use `tensor.contiguous()` after the dimension order has been changed, (i.e. after each `permute`) to save memory and increase speed.)

2.5 Cycle GAN

Source: UC Berkeley Deep Unsupervised Learning Spring 2020
Paper: https://arxiv.org/abs/1703.10593
You can refer to CycleGAN implementations here.
Finish problem "CycleGAN" in `vae_gan_release/VAE_GAN.ipynb`. In this problem, you will implement a CycleGAN that does **unpaired image-to-image translation (domain translation)** from MNIST to Colored-MNIST.

Chapter 3

3D Deep Learning

3.1 Rotation Representations, Conversation (HW1 of ML3D)

Background knowledge: 3D Transformations
Let $p := (1+i)/\sqrt{2}$ and $q := (1+j)/\sqrt{2}$ denote the unit-norm quaternions. Recall that the rotation $M(p)$ is a 90-degree rotation about the X axis, while $M(q)$ is a 90-degree rotation about the Y axis. In the notes, we composed the two rotations $M(p)$ and $M(q)$. Here, we instead investigate the rotation that lies halfway between $M(p)$ and $M(q)$.
The quaternion that lies halfway between p and q is simply

$$\frac{p+q}{2} = \frac{1}{\sqrt{2}} + \frac{i}{2\sqrt{2}} + \frac{j}{2\sqrt{2}}$$

Questions:

1. Calculate the norm $|(p+q)/2|$ of that quaternion, and note that it is not 1. Find a quaternion r that is a scalar multiple of $(p+q)/2$ and that has unit norm, $|r| = 1$, and calculate the rotation matrix $M(r)$. Around what axis does $M(r)$ rotate, and through what angle (say, to the nearest tenth of a degree)?

2. What are the exponential coordinates of p and q?

3. (skew-symmetric representation of rotation) In this problem, we use $[\omega]$ to represent a skew-symmetric matrix constructed from $\omega \in \mathbb{R}^3$ as instructed in class:

   (a) Build the skew-symmetric matrix $[\omega_p]$ of p and $[\omega_q]$ of q, and derive their rotation matrices.

   (b) Using what you have above to verify that the following $\exp([\omega_1] + [\omega_2]) = \exp([\omega_1])\exp([\omega_2])$ relationship does *not* hold for exponential map in general (Note: the condition for this equation to hold is $[\omega_1][\omega_2] = [\omega_2][\omega_1]$). Therefore, composing rotations in skew-symmetric representation should not be done in the above way.

   (c) Given two point clouds $X, Y \in \mathbb{R}^{3\times n}$ sampled from the surfaces of two shapes (two teapots, provided in rotation_conversion.npz), we aim to estimate the rigid transformation to best align them. For simplicity, assume that we have found the correspondence between the two

point clouds, so that the j-th column of X and Y are matched points. We also assume that the two point clouds are already aligned by translation. Then, the point cloud alignment problem can be formulated as the following Stiefel manifold optimization problem:

$$\begin{aligned} \underset{R}{\text{minimize}} \quad & \|RX - Y\|_F^2 \\ \text{subject to} \quad & R^T R = I \\ & \det(R) = 1 \end{aligned}$$

We can solve it using our knowledge of the exponential map. Note that, given a rotation matrix R_1, rotation matrices in its local neighborhood R_2 can be parameterized as $R_2 = R_1 \exp([\Delta\omega]) \approx R_1(I + [\Delta\omega])$ for $\Delta\omega \approx 0$.

i. Use the above knowledge to build an optimization algorithm by filling in the following routine:

Step 1: Initiate $R := I$

Step 2: *Describe your routine to find a* $\Delta\omega$
Hint: Convert the objective to a linear least square problem and solve by your solver in Ch. 1.1:

$$\begin{aligned} \underset{\Delta\omega}{\text{minimize}} \quad & \|A\Delta\omega - B\|_F^2 \\ \text{subject to} \quad & \|\Delta\omega\|^2 \leq \epsilon \end{aligned}$$

Step 3: Update R by $R := R\exp([\Delta\omega])$

Step 4: Go to Step 2

ii. [Programming Assignment] Use your algorithm to solve the point cloud data alignment problem with our provided data.

Note: For the point cloud alignment problem under rigid transformation, there exists a closed form solution to be covered in our next homework. While the closed form solution is more efficient, this iterative solver has better flexibility (e.g, modified derivation may work with different objective and/or constraint set), and is often used in solving non-rigid alignment problems.

4. Double-covering of quaternions

(a) What are the exponential coordinates of $p' = -p$ and $q' = -q$? What do you observe by comparing the exponential coordinates of $(p, -p)$ and $(q, -q)$? Does this relation hold for any quaternion pair $(r, -r)$? If it does, write down the statement and prove it.

(b) Note that the above is called *double-covering* of quaternions. Based on this property of quaternions, answer the following question. When designing a neural network to regress a quaternion output, can you use the L2 distance between the ground truth quaternion and the predicted quaternion? Why and why not?

3.2 Point Cloud Processing (HW1 of ML3D)

In this exercise, we will practice common point cloud processing routines. All problems are programming assignments.

1. Given mesh saddle.obj, sample $100K$ points uniformly on the surface. You may use a library (such as trimesh or open3d) to do it.

2. Use iterative farthest point sampling method to sample $4K$ points from the $100K$ uniform samples. You need to implement this algorithm. You may only use computation libraries such as Numpy and Scipy. *Hint:* To accelerate computation, use Numpy matrix operations whenever possible (they can be 100x times faster than pure python). You may also want to use a progress bar library since the computation may take a minute.

3. At each point of the $4K$ points, estimate the normal vector by Principal Component Analysis using 50 nearest neighbors from the $100K$ uniform points (You can use `sklearn.decomposition.PCA`). Since making the direction of normals consistent is a non-trivial task, for this assignment, you can orient the normals so that they roughly points in the Y direction.

4. Principal curvature estimation for point cloud: Modify the Rusinkiewicz's method (Lecture PDF) and apply it to this point cloud. Describe your algorithm and plot the Gaussian curvature for the shape.

3.3 Shape Deformation (HW2, ML3D)

Background knowledge: Single image to 3D

1. (Laplacian) Given a mesh $M = (V, E, F)$, we assume that the adjacency matrix is $A \in \mathbb{R}^{n\times n}$, $D \in \mathbb{R}^{n\times n}$ is a diagonal matrix where $D[i,i]$ is the degree of the i-th vertex. The **Laplacian** matrix is defined as $L = D - A$.

   Prove that:

   (a) $\sum_{(i,j)\in E} \|x_i - x_j\|^2 = x^T L x$ for $x \in \mathbb{R}^n$.

   (b) $L \in \mathbb{S}^n_+$, i.e., L is a symmetric and positive semi-definite matrix.

   (c) For the data matrix $P \in \mathbb{R}^{n\times 3}$ where each row corresponds to a point in $\mathbb{R}^3$, denote the columns of P as $P = [x, y, z]$ and rows of P as $P = [p_1^T; p_2^T; \ldots; p_n^T]$, show that $\sum_{(i,j)\in E} \|p_i - p_j\|^2 = x^T L x + y^T L y + z^T L z$. (hint: Use the conclusion from 1(a))

2. **Normalized Laplacian** is defined as the normalized version of the Laplacian matrix above:

$$L_{norm} = D^{-1} L \tag{3.1}$$

   (a) Prove that the sum of each row of L_{norm} is 0.

   (b) The difference between a vertex x and the average position of its 1-ring neighborhood is a quantity that provides interesting geometric insight of the shape (see Figure 3.1). It can be shown that,

$$x - \frac{1}{|N(x)|} \sum_{y_i \in N(x)} y_i \approx H\vec{n}\Delta A \tag{3.2}$$

   for a good mesh, where $N(x)$ is the 1-ring neighorhood vertices of x by the mesh topology, $H = \frac{1}{2}(\kappa_{min} + \kappa_{\max})$ is the mean curvature at x (in the sense of the underlying continuous surface being approximated), $\vec{n}$ is the surface normal vector at x, and ΔA is a quantity proportional to the total area of the 1-ring fan (triangles formed by x and vertices along the 1-ring).

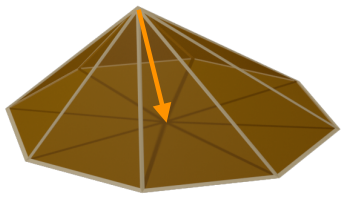

Figure 3.1: Curvature approximation by discrete Laplacian

Define $\Delta p_i := p_i - \frac{1}{|N(p_i)|}\sum_{p_j \in N(p_i)} p_j$. Prove that $\Delta p_i = [L_{norm}P]_i$, where P and p_i are defined as in 1(c), and $[X]_i$ is to access the i-th row of X.

3. (programming) Please load the deform_source.obj and deform_target.obj files using the trimesh library of Python, and optimize to deform the vertices of the `deform_source.obj` to match `deform_target.obj`. Plot the source object, target object, and deformed object.

   (a) Warm-up by Chamfer-only loss: Use the provided Chamfer distance function as a loss to deform the source towards the target. Optimize the position of the vertices in the source mesh by torch (you can just use the Adam optimizer). Show the result and describe what has happened in the deformation process by language.

   (b) Curvature and normal-based loss: We observe that Chamfer loss alone is not able to deform the source mesh properly, and additional loss needs to be added to regularize the process. Here we introduce a simple idea that would work for the provided instances by matching $\{\Delta p_i\}$:

   - First, we compute the Δp_i for each vertex of the source and the target meshes using 2(b).
   - Then, when we compute Chamfer-loss as in 3(a), we actually know about the correspondences across the source and target mesh vertex sets. For each pair of correspondences found in computing the Chamfer loss, we can use the L_2-norm square difference between the corresponding Δp_is' as the loss. Note that this loss computation can be bidirectional (from source to target and from target to source).

   Implement this loss in pytorch and combine it with the Chamfer loss to deform the source shape. Show your final results and deformation process (*e.g.*, the deformed mesh at every 100 steps).

   Note: `deform_source.obj` and `deform_target.obj` have similar amount of vertices, so the ΔA for vertices in the two meshes do not differ much, and matching Δp_is' corresponds to match mean curvatures and normal direction. If there is significant difference in vertex numbers, we need to compensate for the effect caused by significantly different ΔA in the two meshes.

3.4 Pose Estimation (HW2, ML3D)

In the next problems, you will use 2 methods, ICP and PointNet, to predict object poses in a dataset. The training data and testing data are provided at
https://drive.google.com/drive/folders/11TPw-XOypMLEgZLXUwFdBEHBPaMi0VCs?usp=sharing
You can download the data or directly use it in Google Colab by mounting the folder.

1. The task is to predict 6D poses for all the objects of interest in the scene given the image and the depth map. There are 79 different object classes in total. In each scene, there is only one instance for each object class. Across different scenes, instances of an object class might be scaled differently (in fact, only a discrete range, e.g. 0.5 or 1.0). This scale is provided in both training and testing datasets.

2. Here are some notations you might need to know before reading the rest.

   - NUM_OBJECTS: total number of objects we have. it is 79 in this assignment.
   - object_id: each object class is assigned with a unique id, decided by its order in objects.csv.

3. Data description. In training data, there are 2 folders: v2.2 and splits/v2, and there is a file object_v1.csv. The v2.2 folder contains the training data and splits/v2 is a pre-made train/validation split. You are allowed to create your own train/validate split and ignore splits/v2 completely. In splits/v2/train.txt and splits/v2/val.txt, each line is in the format "{level}-{scene}-{variant}". You can use it to find corresponding data files in the v2.2 data folder. Each variant with the same {level}-{scene} uses the same set of object classes with different poses. Here we give a detailed description of all files.

   - **objects.csv**. It provides metadata for the object meshes used to generate the scenes. The important fields are:
     - location: described where to find the meshes in models.zip.
     - geometric_symmetry: describe what symmetrical properties this object has. For example "z2|x2" means this object has a 2 fold symmetry around z axis and a 2 fold symmetry around x axis (which also implies a 2 fold symmetry around y axis). You can see Wikipedia entry Rotational_symmetry for details. "no" means this object has no symmetry; "zinf" means this object has an infinite-fold symmetry around z (e.g. cylinder). The symmetry properties are considered in our evaluation metric (e.g., for a cube, there are 24 rotations that will result in 0 error in evaluation).
     - visual_symmetry: similar to geometric symmetry, but considers object texture. Our evaluation metric will NOT consider visual symmetry, but it is an interesting research topic.
   - **{level}-{scene}-{variant}_color_kinect.png**: an RGB image captured from a camera containing the target objects.
   - **{level}-{scene}-{variant}_depth_kinect.png**: a depth image captured from a camera containing the target objects. The depth is in the unit of mm. You need to convert it into m.
   - **{level}-{scene}-{variant}_label_kinect.png**: a segmentation image captured from a camera containing the target objects. The segmentation id for objects are from 0 to 78.
   - **{level}-{scene}-{variant}_meta.pkl**: camera parameters, object names, gorund-truth poses, etc.
     - poses_world (list): The length is NUM_OBJECTS; a pose is a 4x4 transformation matrix (rotation and translation) for each object in the world frame, or None for non-existing objects.
     - extents (list): The length is NUM_OBJECTS; an extent is a (3,) array, representing the size of each object in its canonical frame (without scaling), or None for non-existing objects. The order is xyz.

  - scales (list): The length is NUM_OBJECTS; a scale is a (3,) array, representing the scale of each object, or None for non-existing objects.
  - object_ids (list): the object ids of interest.
  - object_names (list): the object names of interest.
  - extrinsic: 4x4 transformation matrix, world $\rightarrow$ viewer(opencv)
  - intrinsic: 3x3 matrix, viewer (opencv) $\rightarrow$ image
- We provide a Jupyter notebook to show how to use the data and also some handy scripts for visualization.
- In testing_data, The only difference from training data is that there are no splits and the ground truth poses are not provided in the metadata.
- There is also the models.zip, which provides the meshes and textures for the objects. You will need to sample canonical-space point cloud from the meshes for ICP.

4. Goal. Your goal is to predict the poses of objects in the testing data. The segmentation masks for objects are given so you do not need to solve the detection problem. You can simply segment out the point cloud for each object and use PointNet or ICP to solve the problem. You need to do it first using ICP and then using neural network.

5. Output format. We provide a sample output file "result.json". You need to read and understand the submission format.

6. Evaluation metrics. You should evaluate pose accuracy using the following criteria: success if rotation error < 5 degree and translation error < 1 cm.

7. Report.

   - A short paragraph describing your method and result. (A short version of "our method" and "experiments" section of academic publications) For now, report the success rate on the training set. Visually show a few pose estimation results on the testing set.

3.4.1 ICP-based Pose Estimation

Solve the problem above with an ICP-based method. You will need to sample canonical-space point cloud from the provided object meshes. You do not need training for this part.

3.4.2 Learning-based Pose Estimation

Solve the problem above with a learning-based method. We recommend using PointNet.

3.4.3 Combine 3.4.1 and 3.4.2

Find a way to combine problem 2 and 3 and improve your score. Describe your method in report and show increase of performance.

3.5 3D Semantic Segmentation (HW0, RoboML)

In this problem, you need to train a neural network to solve the 3D shape part segmentation task. Specifically, we focus on the ShapeNet chair category, and there are only four types of parts: arm, back, leg, and seat. The dataset is stored as a folded named “3DSeg" here.
We provide 1,000 annotated shapes as training data. You can find input point clouds in `3DSeg/train/pts` and ground truth labels in `3DSeg/train/label`. For point cloud files(*.pt), each line denotes the 3D coordinate of a point, and the number of points may vary. For label files(*.txt), each line denotes the part annotation of the corresponding point, where 1-4 indicates arm, back, leg, and seat, respectively. Fig. 3.2 shows some examples of the training data.

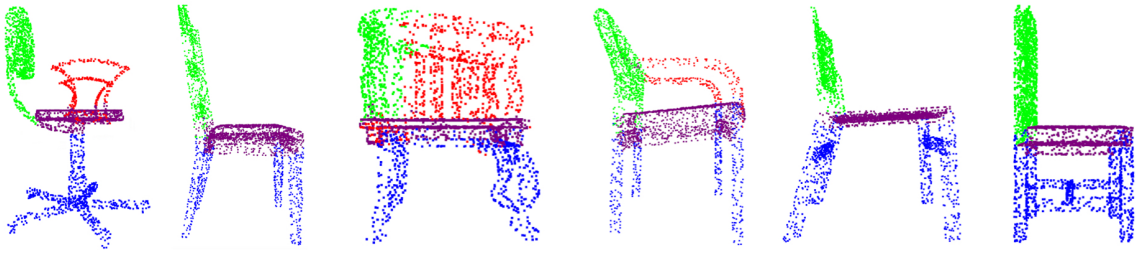

Figure 3.2: Training data: point clouds with ground truth label.

After training, you need to test your model on six testing point clouds provided in 3DSeg/test/ and visualize the results in your report. You can use any 3D library (e.g., Open3D) to visualize the point clouds. **Please color arm parts in red, back parts in green, leg parts in blue, and seat parts in purple**. You can leverage any existing 3D neural network (e.g., PointNet) as your network backbone. Since the segmentation task is relatively easy (only one category and four parts), you don’t need very complex networks, and the network training should converge very fast. You can use quantitative metrics (e.g., `mIoU`) to check your implementation. However, your results will only be evaluated based on visual appearance, and you do not need to achieve very high performance.
Hints:

1 the example implementation of PointNet.

2 the example of using Open3D: building Open3D PointCloud, visualizing pointcloud.

3 building your train-val split to test the generalization performance of your model.

3.6 3D Instance Segmentation (HW3, ML3D)

In the next problems, you will use a network to do object detection/segmentation in a dataset. The training data is the same as before. We additionally added testing for this problem with the name “testing_data_perception.zip”. The dataset link does not change:
https://drive.google.com/drive/folders/11TPw-XOypMLEgZLXUwFdBEHBPaMi0VCs?usp=sharing
You can download the data or directly use it in Google Colab by mounting the folder.

1. The task is to detect/segment objects for all the objects of interest in the scene given the image and the depth map. Essentially, the input is "xxx_color_kinect.png" and "xxx_depth_kinect.png", and you need to output "xxx_label_kinect.png". For dataset description, please refer to the last homework.

2. Model requirements. You are required to use a network with 3D components. If you only use 2D detection/segmentation networks, your maximum score for the report will be 3/5. We recommend implementing Frustum PointNet: https://arxiv.org/abs/1711.08488, as it is probably the simplest and the performance is very strong. You need to describe your method in the report.

3. Output format. The submission to the leaderboard should be a zip file containing 500 images, their names should be "{level}-{scene}-{variant}_label_kinect.png".

4. Evaluation. [5 points]

   Your score is computed as the higher of the absolute and relative score.

   Absolute score

   - 5 points: if you achieve higher than 80% average IOU.
   - 4 points: if you achieve higher than 60% average IOU.
   - 3 points: if you achieve higher than 40% average IOU.
   - 2 points: if you achieve higher than 20% average IOU.
   - 1 points: Make a submission

   Relative score

   - 5 points: rank 1-5.
   - 4 points: rank 6-15.
   - 3 points: rank 16-20.
   - 2 points: rank 21-25.
   - 1 points: the rest.

5. Report. [5 points]

   The report should contain explanation of your network model and experiments. 1-2 pages with pictures should be enough. Things to include: network architecture, experiment setups, training details, validation and test scores, visualizations.

6. Submission. The submission has 3 parts

   - A short report describing your method and result. (A short version of "our method" and "experiments" section of academic publications. 1 page should be enough, but you may write more if you have interesting findings.)
   - A .zip archive containing your code.
   - You need to submit your results on testing data to our internal benchmark.
     https://storage1.ucsd.edu/cse291ibenchmark/benchmark2
     The benchmark will be online shortly after the late due date of HW2.
     A benchmark submission will take about 20s, please be patient and please avoid trying to submit multiple times.

7. The benchmark suite (IOU) is uploaded to piazza resources “benchmark_pose_and_detection.zip”.

3.7 Point Cloud GAN

Overview “GANs are a framework for teaching a DL model to capture the training data’s distribution so we can generate new data from that same distribution. GANs were invented by Ian Goodfellow in 2014 and first described in the paper Generative Adversarial Nets." GANs are widely studied for 2D image generation. PyTorch provides a tutorial for DCGAN, which is a classical CNN architecture for 2D GANs. Wasserstein GAN (WGAN) proposes an alternative to traditional GAN training. WGAN-GP improves training of WGAN. MMgeneration provides implementations for many 2D GANs.
However, there are fewer works about 3D GANs, especially 3D point cloud GANs. Examples are Learning Representations and Generative Models for 3D Point Clouds [1], Point cloud gan [2], and Rethinking Sampling in 3D Point Cloud Generative Adversarial Networks [3].

Problem In this problem, you are asked to train a point cloud GAN on a pre-processed subset of ShapeNet. The data can be downloaded here from this repo. You need to first implement and train a basic (unconditioned) GAN on the data. Then, you need to implement and train a WGAN-GP on the same data.
The training pipeline (like data flow and losses) is almost the same for both image and point cloud GANs, except that the discriminator and the generator need to deal with point clouds now. You can follow [3] to use PointNet as the discriminator, and use a simple MLP as the generator. The output of the generator should be point clouds rather than latent codes in [1]. For WGAN-GP, you need to think over how to interpolate between two point clouds.
Apart from training generative models on point clouds, you need to:

- Visualize some generated point clouds (at least 16 examples, maybe organized in a 4x4 grid), including both success and failure cases (if any). To visualize point clouds, you can try Open3D, trimesh, matplotlib or even Blender.

- For each generated point cloud, find the most similar shape in the "real" data. The metric to measure similarity can be any distance function on point clouds, like Chamfer Distance or Earth Mover Distance. It is suggested to compare and visualize them side by side.

3.8 Learning-based MVS

Overview Reconstructing 3D geometry from images is a long standing problem in computer vision. With its applications range from navigation, autonomous driving to AR / VR. The problem has recently caught significant attention from these relevant communities. Among all 3d reconstruction methods, the multi-view stereo (MVS) aims at solving for the depth map from a set of neighboring images accompanied by camera intrinsic and extrinsic parameters. MVS methods rely on parallax to estimate scene geometry. A special case where only two views are given is typically referred to as stereo matching. We suggest you to go over OpenCV Depth Map from Stereo Images Tutorial, DispNet, PSMNet, and MVSNet before solving this problem. The Multi-View Stereo: A Tutorial could serve as a great resource to look up for some details.

Problem Implement a MVSNet on your own. You only need to implement the basic feature extraction, cost volume building, cost aggregation, and depth generation pipeline. You do not need to implement RefineNet and other techniques in the paper. You implementation should take 3 images at a time, and generate depth maps with the same sizes as input images. Train your model on the DTU dataset. You could consult to public repositories, but not allowed to copy their code directly. You are allowed to use pre-processed datasets available online.

1. Evaluate your method on the bottles dataset. Plot your depth map for image id `1_val_0026`, `1_val_0028` and `1_val_0073` with a proper color map. You can select whichever input views you need (no matter marked `train` or `val`) in this problem.

2. Evaluate your method on all 200 images in the dataset. Convert these depth maps into a complete clean point cloud of the object.
   Hint: You should find way to filter incorrect points before fusing. Consult to the MVSNet paper for a possible way to achieve this.

3. What are the major issues you observed with MVSNet? Do you observe any failure cases during training or evaluation? Discuss any findings you have here and propose solutions if possible.

3.9 Neural Radiance Field

Overview The neural radiance field (NeRF) is a very recent method for scene representation. Unlike most explicit scene representations relying on meshes or voxels, neural radiance field encode all information explicitly using a multi-layer perceptron (MLP), which yields to smooth and continuous representation of scenes. The NeRF is most commonly used to generate photo-realistic images. Given a set of posed images for a same scene, we can fit a NeRF model for that scene and later use the model for synthesizing novel views. You should refer to the original NeRF paper for more details.

Problem

1. Train a NeRF model using all images in the bottles dataset. Synthesize image id `2_test_0000`, `2_test_0016`, `2_test_0055`, `2_test_0093`, and `2_test_0160` with your model. The camera poses of these views are given in the `pose` folder. You can select as many training views you need (no matter marked `train` or `val`) in this problem.

2. Train a NeRF model using images in the indoor dataset. Synthesize image id `164`, `180`, and `196` with your model. The camera poses of these views are given in the `poses` folder.

3. Is it possible to extract depth from a learned NeRF model? Try to generate explicit geometry representations (meshes or point clouds) from the above models you have. For the Lego dataset, how is the geometry quality compared to the one you have in the MVSNet experiment?

4. What are the major issues you observed with NeRF? Do you observe any failure cases during evaluation? Discuss any findings you have here and propose solutions if possible.