HW4

B08505048 工海四 劉名凱

Problem 1:

(1)

a.

Nerf represents a scene as a 3D coordinate (x, y, z), each location has its own volume density and view-dependent color. When these two properties are provided (predicted), we can use mathematical method (classical volume rendering) to render the color of rays from different angle passing through the scene. Nerf encodes function which inputs are ((x, y, z), (θ , ϕ)) and which output are (σ , c) into MLP, so that Nerf can predict volume density and color in different location according to camera angles.

b.

I think adding positional encoding is the most important part in Nerf. From the Fig.4 in original Nerf paper we can observe that without positional encoding, Nerf can't synthesize a high-quality viewpoint, minor details are nixed together. Only with positional encoding can MLP successfully learn a high frequency variation in color and geometry.

c.

Pros:

1. Nerf encodes **continuous** volume into the MLP, which enable it to produce high-quality renderings compared to other method like SRN which can only represent low-frequency geometry.
2. Nerf only requires moderate number of input images to fit a scene, unlike previous works
3. Nerf's weight only requires small memory space.

Cons:

1. Training process and rendering speed of Nerf are slow compared to other method such as LLFF which use 3D convolutional network or other mesh-based method.

(2)

According to original DVGO paper *Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction*, author represents the scene as voxel-grids, each grid cells store the target information (e.g., density, color). During coarse stage sampling, model will find the coarse 3D areas of interest (where objects are) and predict the volume density. In later fine stage, model can only focus on areas of interest and thus greatly reduce the number of quired points, which effectively reduce the time to render images. During fine stage, model predicts features based on location. Features will be fed into MLP and generate view-dependent color emission.

I tried several settings including changing iteration, number of voxels in coarse or fine stage. However, there are no significant improvement on the validation set. Increasing iteration or number of voxels will only increase the performance on training set. Therefore, in my final setting, I only increase the number of voxels in fine stage, which I expected model benefits from it by being able to predict more detailed result.

(3)
PSNR: Representing the distance between ground truth image and produced image. Because PSNR is proportional to the reciprocal of Mean Square Error between two images, the higher the PSNR is, the more similar two images are.

SSIM: Comparing luminance, contrast and structure between two images. The higher the SSIM is, the more similar two images are. Luminance, contrast and structure can be derived from mean, standard deviation of input region.

LPIPS: Feed two images into pretrained neural network and compare the distance between respective output feature. LPIPS yields better results than low-level metrics such as PSNR and SSIM, it can better represent difference between two images from human's perspective.

Validation set

	PSNR	SSIM	LPIPS (vgg)	LPIPS (alex)
Set1	35.2431	0.9747	0.0404	0.0220
Set2	35.3213	0.9757	0.0377	0.0185

Training set

	PSNR	SSIM	LPIPS (vgg)	LPIPS (alex)
Set1	39.5097	0.9854	0.0251	0.0120
Set2	41.5027	0.9904	0.01675	0.0059

Set1:

coarse training iteration:	10000 (default 5000)
coarse model number of voxel:	2048000 (default 1024000)
coarse model number of base voxel:	2048000 (default 1024000)
otherwise:	default

Set2:

coarse training iteration:	10000 (default 5000)
coarse model number of voxel:	2048000 (default 1024000)
coarse model number of base voxel:	2048000 (default 1024000)
fine model number of voxel:	320^3 (default 160^3)
fine model number of base voxel:	320^3 (default 160^3)
otherwise:	default

The difference between set1 and set2 are number of voxels in both coarse and fine model. I increase the number of voxels to improve the quality of rendered images. However, there is no significant improvement on validation set. I guess the reason is that set1 is already good enough, increase number of voxels won't make much changes.

Problem 2:

(1)

SSL method: BYOL (Bootstrap your own latent)

Data augmentation: I follow the data augmentation in official github

1. Color Jittering
2. Random Grayscale
3. Random Horizontal Flip
4. Gaussian Blur
5. Random Resized Crop
6. Normalize (mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))

Learning rate scheduling: Cosine Annealing Schedule (follow the original paper)

Optimizer: Adam(learning rate = 3e-4)

Batch size: 256 (larger batch size won't fit in the GPU memory)

Warm up epoch: 10

All training settings except learning rate and batch size are same as those proposed in original paper. At first, I trained for over 200 epochs. However, when fine tuning on office dataset, the backbone only achieved 25% accuracy. After some simple monitoring on the SSL training process, I observed that loss was very unstable, for example, loss in epoch 150 could be larger than epoch 1. Therefore, I saved backbone weight every ten epochs and fine tune with the version that had lowest loss, which led to a better result.

(2)
Setting A: 0.2118
Setting B: 0.4852
Setting C: 0.5
Setting D: 0.3448
Setting E: 0.3399

Among five settings, model w/o SSL pre-training yields the worst result, which indicates that SSL pre-training is indeed beneficial and effective for classification accuracy. SSL pre-training can greatly improve model quality even when pre-training dataset (Mini-ImageNet) has little similarity with downstream dataset (Office-Home), not to mention when pre-training dataset and downstream dataset are similar.

Accuracy of setting D, E are much lower than setting B, C. I believe this can be attributed to the fact that model does not see images related to Office when pre-training. Therefore, training only classifier is not enough. If a higher accuracy is desired, we must train full model.