Implicit Zoo: A Large-Scale Dataset of Neural Implicit Functions for 2D and 3D Scene Supplementary Material

Qi Ma[1,2] **Danda Pani Paudel**[2] **Ender Konukoglu**[1] **Luc Van Gool**[1,2]
[1]Computer Vision Lab, ETH Zurich [2]INSAIT, Sofia University

In this supplementary material, we first detail the data generation process in Section 1 and provide more information on the implementation of the learnable tokenizer in Section 2. Next, we present additional details and experimental results on the benchmarks in Section 3. Finally, we provide information needed in checklist in Section 4. To gain a better understanding of our dataset and proposed benchmarks, please refer to the introductory video in the supplementary materials or the one available on our project page, which offers an overview of our dataset and its applications.

1 Additional details of Dataset Generation

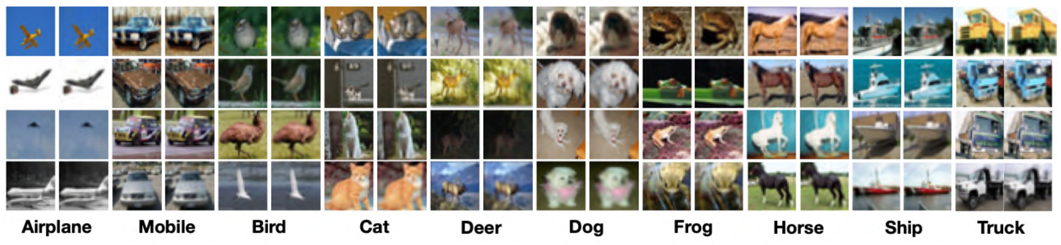

Figure 1: **Additional examples on CIFAR-10-INRs dataset** We present additional data examples, where the left side of each image pair shows the ground truth and the right side displays the results queried from the INRs.

Speed up Training In [1], the authors propose meta-learning and implicit function modulation to accelerate the training process. Similarly, [2] and [3] reduce training time by employing smaller models and optimizing the number of iterations. In our 2D dataset, we observed that normalizing images before training implicit functions significantly speeds up training iterations. Therefore, we chose to normalize images and get rid of Sigmoid activation function in the final layer. In the 3D cases, we use a small model with 4 layers and a width of 128, without any skip connections. During training to enhance the performance with limited iterations, we propose an adaptive sampling method that focuses more on rays corresponding to 2D RGB values that are not white (likely to be the background). This approach is particularly beneficial for handling light-colored cases and tiny objects as shown in Fig 4. We observed that the training loss converged at 20k steps with learning rate 5e-4.

More Examples of data We provide more examples of data on 1, 2, 1, 4. Note that if you zoom in, you may notice some artifacts in the CIFAR-10 dataset. For example, in the bottom row of the dog category in Fig 1 , the dogs appear slightly blurry. For a 32x32 image, a PSNR of 30 results in more noticeable visual differences compared to larger images, as seen in Fig 2. To address this issue, we refined the CIFAR-10 data as described in the main paper, increasing the average PSNR to approximately 35 and resulting in a smaller standard deviation across different classes, as shown in

Fig 9. Additional experiments on the refined dataset, reported in Table 2, align with the findings in the main paper.

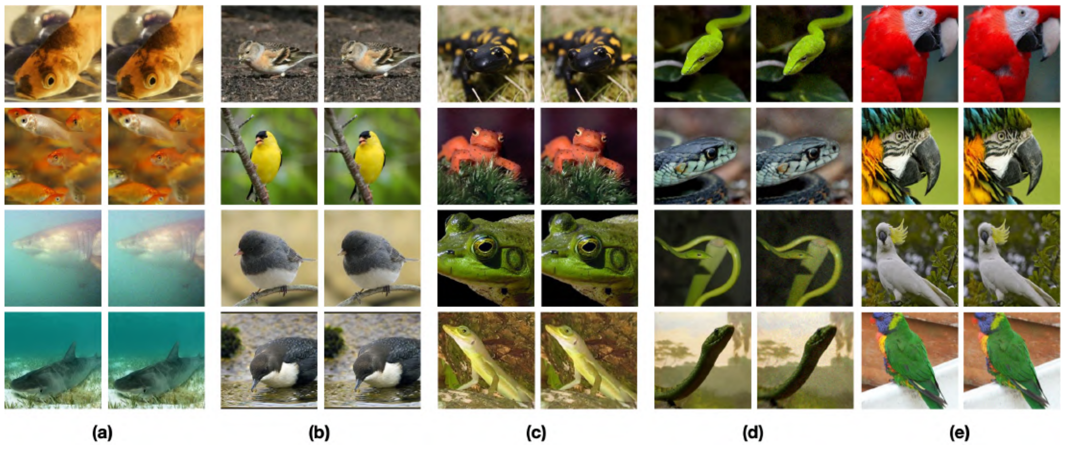

Figure 2: **Additional examples on ImageNet dataset** We present additional animal images from the ImageNet dataset, which is one of the motivations behind naming this work *Implicit Zoo*. Comparing with the ground truth images on the left, the reconstructions are of very high quality.

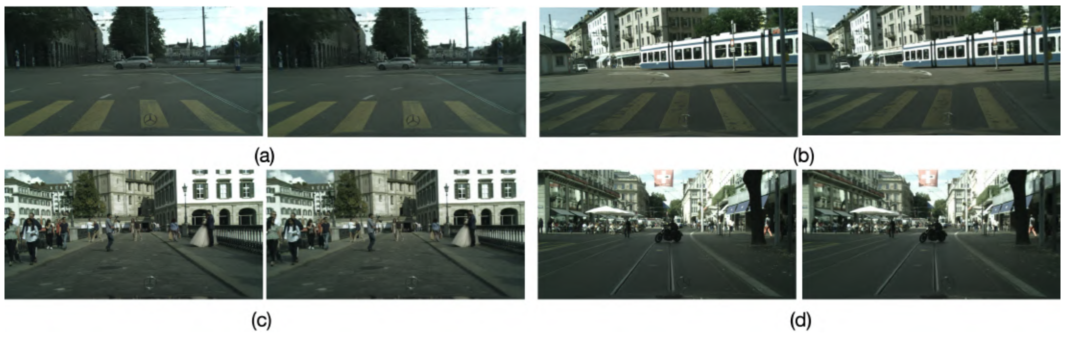

Figure 3: **Additional examples on Cityscapes dataset** We present additional data samples from Cityscapes-INRs. Notably, fine details such as pedestrians in (c) and significant illumination changes in (d) are well-preserved in the reconstructions.

Data statistics We report PSNR across classes of dataset in Fig 9, 10. For Cityscapes-INRs results please refer to main page. Note that the PSNR differences in 2D cases are minimal due to the quality control and further refinement we implemented. Some of the classes in ImageNet results are higher than others, indicating better performance achieved during the initial phase of training.

Scene filtering As shown in Fig 11 the rendering PSNR for novel view changes a lot cross different classes (180 claases) with standard deviation 3.87. This is mainly because the various objects the dataset include. As shown in Fig 4, we observe that PSNR tends to be higher for light-colored objects because their colors align with the white-background assumption [4]. A similar trend is observed for small objects. During data filtering, we first exclude cases with a PSNR below 25. For classes with fewer than five scenes, we ignore the entire scene in this class. Ultimately, we retain 5,287 valid scenes for our experiments.

Data releasing. We uploaded CIFAR-10-INRs, ImageNet-10-INRs Omniobject3D to Kaggle and can be found in project page. The Cityscapes-INRs dataset will be released shortly on the Cityscapes team's official after this paper is published publicly. Additionally, we are working on a refined version of ImageNet with PSNR > 35 and training a larger NeRF model on Omniobject3D, utilizing a coarse and fine model with 8 layers and a width of 256. The benchmark code will also be released on the above webpage.

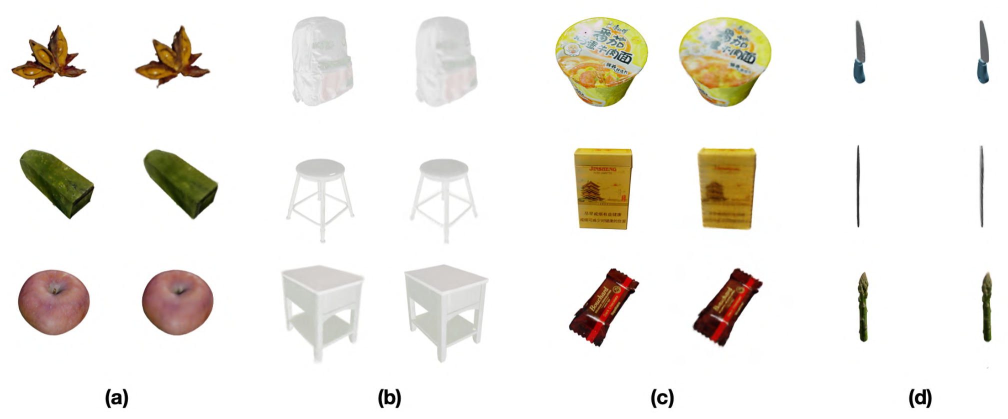

Figure 4: **Additional examples on Omniobject3D dataset** We present additional examples on Omniobjecct3D. We observe that when objects are large, have rich colors, and relatively simple surfaces, our reconstruction performs very well (a). However, in more challenging cases such as (b) shallow-colored objects, (c) complex surfaces with text information, and (d) small or thin objects, the reconstruction quality is less satisfactory.

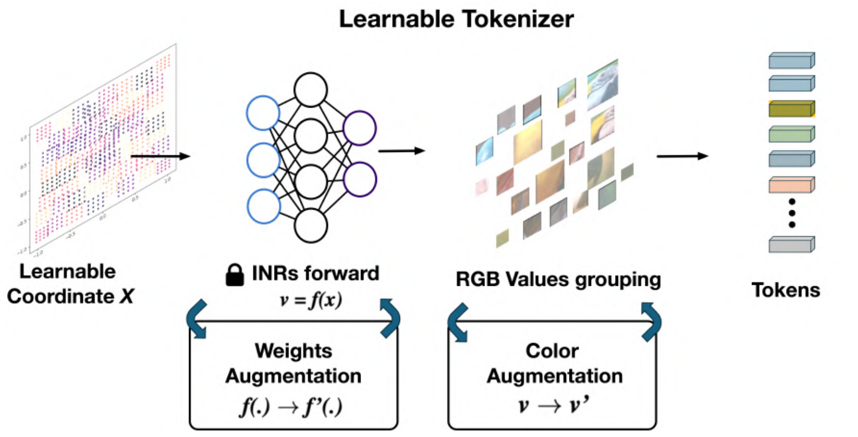

Figure 5: **Differential Augmentation** We propose geometric augmentation in weights-space and color augmentation in RGB-space.Following [5] we propose 15 differentiable transforms which enhance our dataset application.

2 Additional details of Learnable tokenizer

2D implementation We provide more detailed information on learnable tokenizer and different RGB grouping on 2D implementation here. We first map the coordinates to $(-1,1)$ and then divide them uniformly with patch size P to N patches, each containing P^2 coordinates. We calculate the center coordinate c_i for each patch $i \in \{1, 2, ...N\}$ and determine the coordinate difference of each coordinates to the center coordinate d_{ij}, where $j \in \{1, 2, ...P^2\}$. Thus, all coordinates x_{ij} can expressed as $x_{ij} = c_i + d_{ij}$. For (b) Learnable Scaling, we introduce a learnable scaling factor s_i for each patch. The queried coordinate is then given by $x'_{ij} = c_i + s_i d_{ij}$. For (c) Learnable Centers we make c_i themselves also learnable. Both method (b) and (c) keep the grid shape. For (d) Learnable pixels instead of learning a coordinate difference we directly make all coordinate $x_i j$ learnable. Finally in (d) we divide at beginning the coordinates randomly.Furthermore, to stabilize the training and ensure the learnable scale remains non-negative and the learnable pixels stay within the range $(-1,1)$ a extra $Tanh(.)$ activation is applied on scaling factor and $Sin(.)$ is added on learnable coordaintes.

Augmentation	Rotate	Translate	ShearX
Implementation	$W_t = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}$	$b_t = W \triangle b$	$W_t = \begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix}, b_t = W \begin{bmatrix} s \\ 0 \end{bmatrix}$

Table 1: **Implantation of geometric augmentation on weight-space**: We implement geometric transformation by modifying the weight W and bias b in first layer of INRs

Differential augmentation As discussed in the main paper, differential augmentation is crucial to ensure that gradients can backpropagate to the learnable tokenizer. Unlike previous works [6, 7] hat train more INRs on augmented data, we implement [5] in a differential manner. Note that we propose to implement non-differential geometric augmentations in weight-space to make them differentiable. As shown in Fig 5, we first implement geometric augmentation such as Rotate, ShearX, ShearY, TranslateX, TranslateY, Cutout in by modifying the first layer of INRs. Specifically, we adjust the weights and biases as $W' = W + W_t$ and $b' = b + b_t$. This is calculated by $W'(x') + b' = Wx + b$ where x' is the corresponding coordinates after transformation.

Next, we implement color augmentations such as AutoContrast, Equalize, Solarize, Color Balance, Invert, Contrast, Brightness, and Sharpness in RGB space. Two main challenges arise: first, some color augmentations are non-differentiable, such as the Equalize operation, which creates a uniform distribution of grayscale values in the output image, and the Posterize operation, which reduces the number of bits for each color channel. To address that we first calculate the transform T outcome of these two operation and add the residual to our rgb value. $\triangle v = T(v) - v$ and $v' = v + \triangle v$. Note that we do not apply this residual addition to all color transformations because we want proposed learnable tokenizer to remain learn from color augmentations. Secondly some out-of-range RGB values can appear due to geometric augmentations $x' \notin (-1, 1)$, as illustrated in the Fig 6. These values can significantly impact downstream tasks such as segmentation and can also interfere with RGB augmentations process. To address this, we propose applying the same geometric transformations to a binary mask M with zero padding for out-of-range values. Before performing Color augmentations, we first apply the mask operation $v_{mask} = Mv$. We adopt some operations from [8].

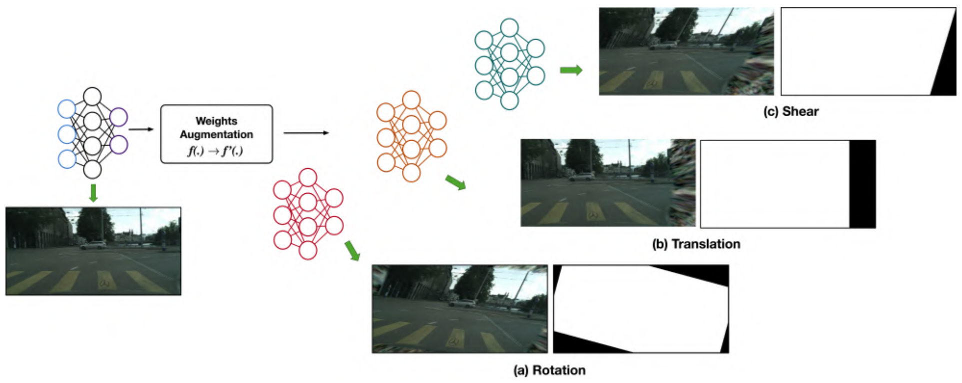

Figure 6: **Masking out-of-range values**: We implement zero-padding by applying same geometric transformation for a binary mask and we show examples for (a)rotation, (b)translation and (c)shear operation. Out-of-range RGB values are clearly visible in the bottom right corner of the transformed image.

3D implementation To lift up the learnable tokenizer to 3D volume tokenization we make following modification. Firstly we uniformly divide the space to N volumes and each include P^3 3D sample points. Then similarly for Learnable Centers + Learnable Scale we calculate the center points c_i for each volume $i \in \{1, 2, ...N\}$. The remaining operations are similar to 2D process.

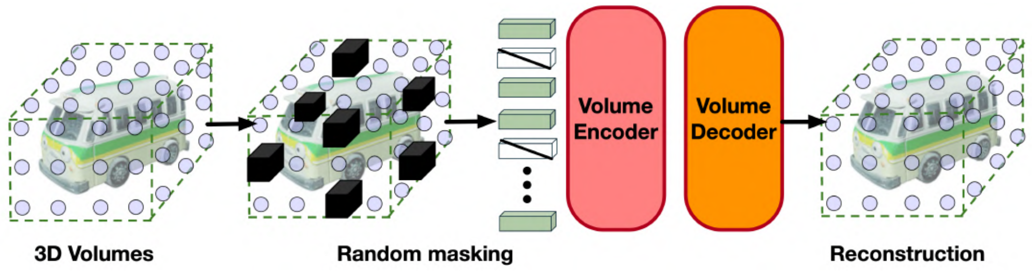

Figure 7: **Illustration of Proposed 3D Volume Encoder pretraining mechanism**We randomly mask out 80% of the volume tokens, allowing the encoder to operate only on the visible tokens. A small decoder processes the full set of encoded visible patches and masked tokens to reconstruct the input color volume and density volume. For qualitative results, please refer to Fig 8

3 Additional details of Benchmark

We provide detailed information on the implementation of our experiments and benchmarks. Additionally, we introduce our proposed pretraining mechanism, which has proven to be effective, as demonstrated in the main paper.

3.1 Implementation details

CIFAR-10-INRs We use ViT-tiny [9] with a depth of 12 and 3 heads for multi-head self-attention. The patch size is set to 4, and we employ a convolution layer with a kernel size of 4 as the embedding layer. For optimization we use Adam optimizer [10] with learning rate is 0.0001 for classify model and learnable tokenizer. Additionally, we apply CosineAnnealing leraning rate scheduler [11]. For regularization,we use regularize weight of $w_{reg} = 1$.

ImageNet-100-INRs: We utilize ViT-base [12, 9] with pretrained model 21k-1k and fine-tuned on our ImageNet-100 dataset.Our experiments are conducted with a batch size of 32, using distributed training on GeForce RTX 4090 GPUs. We employ the AdamW optimizer [13] with a learning rate of 1e-5 for the main model and the learnable tokenizer. A WarmupCosineAnnealingLR scheduler with one warmup epoch is used for learning rate adjustment.

Cityscapes-INRs: Similar to above we use AdamW optimizer [13] with learning rate 1e-4 for segmentation model and 1e-5 for learnable token. This is because in segmentation task we do not want misalignment between supervision area and tokenizaiton area too large. Following [14] we use PolynomialLR scheduler with power of 1.0.

Omniobject3D-INRs: We first introduce the pretraining mechanism, by following [15] we random masking the volume tokens as shown in Fig 7. The volume encoder operates only on the unmasked tokens. The proposed volume encoder operates only on the unmasked tokens and consists of 12 layers, each with 3 heads and an embedding feature dimension of 192. For the decoding process, we utilize 8 layers transformer-based decoder with same head numbers and embedding dimension as encoder. We use a shared and learnable masked token to fill in the originally masked-out positions and apply the positional encoding of the original tokens. The decoder then predicts the original RGB and density values, using mean squared error (MSE) as the loss function.

Next, we use the pretrained encoder for INRs pose regression, demonstrating its effectiveness with improved results across all proposed learnable tokenizers. We train the model for 100 epochs with a batch size of 8, where each batch includes 24 sampled views of a given scene. For non-pretrained experiments, we use a learning rate of 1e-4, while for the pretrained volume encoder, we use a learning rate of 1e-5.

3.2 Additional experiments

We conducted weight-space-only experiments and additional experiments on refined CIFAR-10-INRs, training for 500 epochs. We selected DWSNet [16] and HyperRepresentation [17] for benchmarking, as they represent two primary approaches: one proposes a permutation-invariant network structure to process trained INRs, while the other learn strong encoder to tokenizes the network weights to latent feature.

For DWSNet, we used a 4-layer model with a hidden dimension of 64. For HyperRepresentation, we employed an 8-layer transformer with a hidden dimension of 512. Notably, DWSNet performed poorly on the CIFAR-10 experiments, likely due to the lack of additional information from the input coordinate domain for the RGB 3D higher dimension output [16]. HyperRepresentation performed better but still yielded unsatisfactory results compared to other RGB space-based methods.

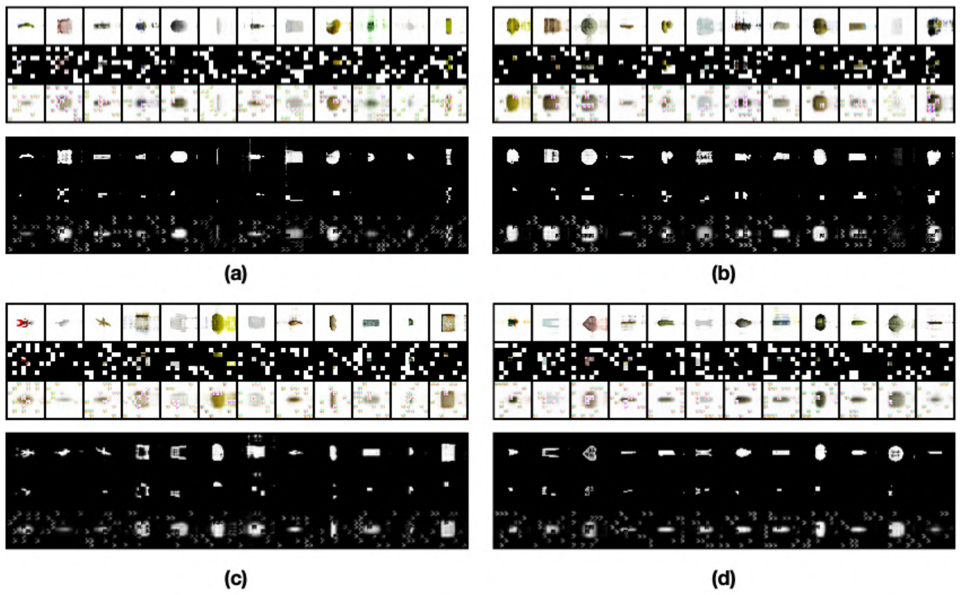

Figure 8: **Visualization on validation set of Omniobject3D reconstruction** We present four batches of reconstruction results in panels (a), (b), (c), and (d). The top row displays the input RGB volumes and density volumes. The middle row shows the masked volumes, while the bottom row illustrates our reconstruction results. Each sample contains 32x32x32 sampled points, and with a volume size of 4, it generates 512 tokens, of which only 102 are visible. Note that the output in known patches location may exhibit some artifacts.

Method	Acc ↑	Precision ↑	F1 ↑
ViT[12]	84.28±0.41%	84.17± 0.44 %	84.25± 0.42 %
ViT[12] + LC	85.11± 0.33%	85.03± 0.38%	85.09± 0.34%
ViT[12] + LP + Reg	**85.35± 0.35%**	**85.33± 0.37%**	**85.34 ± 0.35%**
DWSNet[16]	38.12±1.32%	36.33±1.54%	37.11±1.33%
Hyper[17]	63.14± 1.12%	61.22± 1.45%	62.45± 1.34%

Table 2: **Refined CIFAR-10-INRs Classification.** We conduct additional experiments using refined CIFAR with 500 epochs.We report results from the weight-space-only method and observe that a performance gap still exists between the weights-only method and the RGB-based image method.

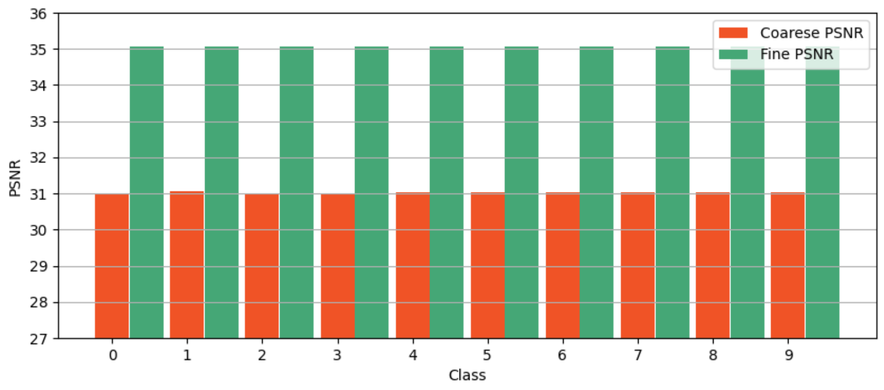

Figure 9: **PSNR for each class in CIFAR-10-INRs** We report the PSNR for all classes of CIFAR-10 INRs. Before refinement, the standard deviation across different tasks is 0.013, and after refinement, it is 0.005.

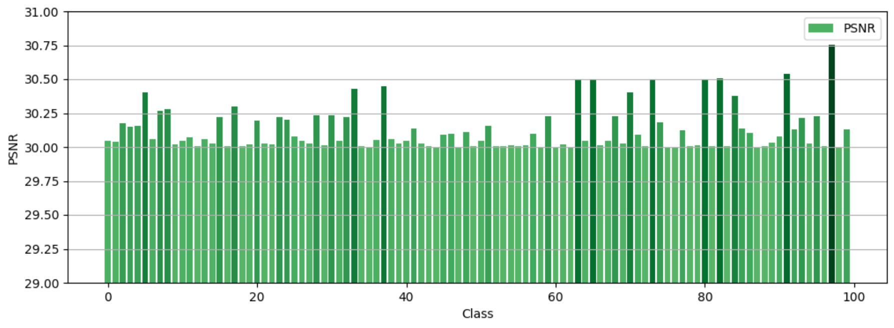

Figure 10: **PSNR for each class in ImageNet-100-INRs** We report PSNR on ImageNet-100 with standard deviation 0.157.

4 Additional information for Checklist

Potential negative societal impacts While our work on proposing a large-scale INRs dataset for 2D and 3D tasks offers significant advancements in the field of implicit neural representations, it is important to consider potential negative societal impacts such like (1) Privacy Concerns: The proposed dataset use other popular public dataset and share the same risk for privacy violations of other dataset. (2) Our work limit only to natural images and more diverse modality should be considered. (3) Environmental Impact: Training large-scale INRs models requires significant computational resources, which can contribute to high energy consumption and increased carbon footprint. This environmental impact is a growing concern with the proliferation of large-scale AI models.

References

[1] Emilien Dupont, Hyunjik Kim, SM Eslami, Danilo Rezende, and Dan Rosenbaum. From data to functa: Your data point is a function and you can treat it like one. *arXiv preprint arXiv:2201.12204*, 2022.

[2] Ziya Erkoç, Fangchang Ma, Qi Shan, Matthias Nießner, and Angela Dai. Hyperdiffusion: Generating implicit neural fields with weight-space diffusion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14300–14310, 2023.

[3] Luca De Luigi, Adriano Cardace, Riccardo Spezialetti, Pierluigi Zama Ramirez, Samuele Salti, and Luigi Di Stefano. Deep learning on implicit neural representations of shapes, 2023.

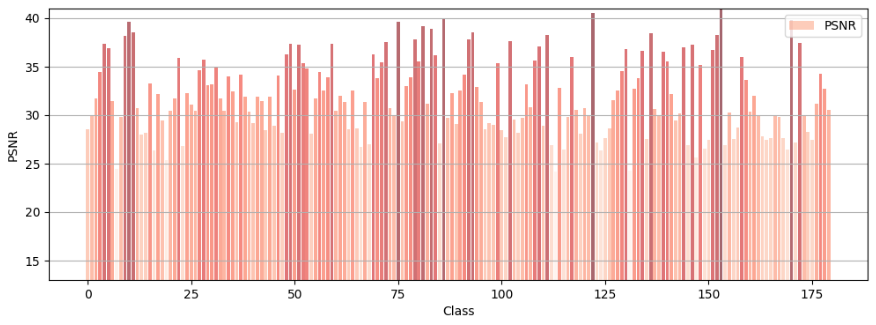

Figure 11: **PSNR for each class in Omniobject3D-INRs-INRs** We report PSNR on Omniobject3D with standard deviation 3.87.

[4] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[5] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space, 2019.

[6] Allan Zhou, Kaien Yang, Yiding Jiang, Kaylee Burns, Winnie Xu, Samuel Sokota, J Zico Kolter, and Chelsea Finn. Neural functional transformers. *Advances in Neural Information Processing Systems*, 36, 2024.

[7] Allan Zhou, Kaien Yang, Kaylee Burns, Adriano Cardace, Yiding Jiang, Samuel Sokota, J. Zico Kolter, and Chelsea Finn. Permutation equivariant neural functionals, 2023.

[8] D. Ponsa E. Rublee E. Riba, D. Mishkin and G. Bradski. Kornia: an open source differentiable computer vision library for pytorch. In *Winter Conference on Applications of Computer Vision*, 2020.

[9] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[11] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017.

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[13] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[14] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers, 2021.

[15] Christoph Feichtenhofer, Yanghao Li, Kaiming He, et al. Masked autoencoders as spatiotemporal learners. *Advances in neural information processing systems*, 35:35946–35958, 2022.

[16] Aviv Navon, Aviv Shamsian, Idan Achituve, Ethan Fetaya, Gal Chechik, and Haggai Maron. Equivariant architectures for learning in deep weight spaces, 2023.

[17] Konstantin Schürholt, Dimche Kostadinov, and Damian Borth. Hyper-representations: Self-supervised representation learning on neural network weights for model characteristic prediction, 2022.