Supplementary Material for iNeMo: Incremental Neural Mesh Models for Robust Class-Incremental Learning

In the following, we provide further details and ablation studies for our paper. In the first section we define the conventions. We then provide the non-incremental performance of both considered representations (R50 and NeMo) as a reference. Afterwards, we show the advantage of considering uncertainty for the classification in Section C and then give a conclusive ablation study over all components of our method in Section D. Since NeMo is trained with additional pose labels that were not available to baselines, we provide an additional study in Section F where we show that pose labels do not improve the baselines. Finally, we conclude with additional details on the background model and pose estimation, as well as all the considered hyperparameters in our method.

A Conventions

In the tables of the main paper, we followed previous work [5, 6, 8] and reported the average of the testing accuracies over all tasks $\mathcal{T}^i$ with Φ_i, which we denoted as $acc(1:i)$. In the supplemental material, we deviate from this setting and **report the final accuracy with $\Phi_{N_{\text{task}}}$ on the whole test dataset after integrating all tasks**, as it determines the final performance loss that is usually most significant. We denote the final accuracy on all seen classes after training on the final task $\mathcal{T}^{N_{\text{task}}}$ as $\overline{acc}(1:N_{\text{task}})$.

B Non-Incremental Upper Bounds

To determine an upper bound for the performance of ResNet50 and NeMo approaches, we train on all classes jointly and provide the results in Table 1. While both approaches are able to achieve similar performance for classification on P3D, NeMo significantly outperforms the RestNet50 on O3D. We suspect that the reason for this is that O3D contains a large number of occluded and truncated objects. NeMo generally outperforms the ResNet50 for pose estimation implemented following [15].

C Considering Uncertainty in Classification

We proposed an extension to the classification strategy introduced by Jesslen *et al.* [4] in Equation 14 of the main paper which was motivated by the observation that classes sharing visually similar features were confused more often when training the model in an incremental setting. We believe that when training on all classes jointly, the contrastive loss between all features of different classes is sufficient to ensure that all parts of different objects have distinct feature representations. However, such disentanglement is significantly more challenging in an incremental setting. The results from Table 2 show that our proposed strategy to exclude pixels that ambiguously relate to meshes of multiple possible classes (i.e. uncertain pixels) brings a significant improvement.

Table 1: We trained the RestNet50 and NeMo approaches jointly on all classes to determine an upper bound for their performance. All networks were initialized with weights from DINOv1 [2], which itself was trained in an unsupervised fashion. For joint training of NeMo, we follow the training protocol of Jesslen *et al.* [4] with the exception of using pre-trained weights as mentioned.

Metric	Type	Repr.	*P3D*	*O3D*
Classification	Joint	R50	98.32	76.23
$\overline{acc}(1 : N_{\text{task}})$ in % ↑	Joint	NeMo	99.27	85.28

Metric	Type	Repr.	*P3D*
Pose $\pi/6$	Joint	R50	74.6
$\overline{acc}(1 : N_{\text{task}})$ in % ↑	Joint	NeMo	87.25
Pose $\pi/18$	Joint	R50	36.5
$\overline{acc}(1 : N_{\text{task}})$ in % ↑	Joint	NeMo	65.81

Table 2: We compare our inference approach to the one proposed by Jesslen *et al.* [4]. When training jointly, the performance is nearly identical. However, when training incrementally, disentangling visually similar features becomes more challenging and our proposed strategy significantly improves the result.

Metric	Type	Inference	*P3D*	*O3D*
			$B0 + 3$	$B0 + 20$
Classification	Joint	[4]	99.28	85.28
	Joint	Ours	99.27	85.28
$\overline{acc}(1 : N_{\text{task}})$ in % ↑	Incremental	[4]	95.06	75.8
	Incremental	Ours	96.41	80.17

Table 3: Top: We provide an ablation study for the 2D ResNet50 and NeMo with the traditional class-incremental techniques LwF and iCaRL. As visible, traditional techniques work less well on NeMo. **Bottom:** We provide an ablation study of our model components and show that all of our additions increase the performance. We indicate the used exemplar selection strategy in the column "Replay", where H denotes the herding strategy [8] and PA our pose-aware exemplar selection strategy. Note that we used our improved inference method from Section C for all methods.

Metric	Method	Repr.	Replay	Init	Pos.	KD	*P3D*	*O3D*
							$B0+3$	$B0+20$
Classification	Finetune	NeMo	-				17.47	17.81
	LwF	R50	-			✓	83.75	56.44
	LwF	NeMo	-			✓	17.45	17.72
	iCaRL	R50	H			✓	91.79	61.75
	iCaRL	NeMo	H			✓	93.72	68.87
$\overline{acc}(1:N_{\text{task}})$ in % ↑	Ours	NeMo	H				93.60	69.01
	Ours	NeMo	PA				94.70	70.32
	Ours	NeMo	PA	✓			94.78	71.67
	Ours	NeMo	PA	✓	✓		94.98	72.09
	Ours	NeMo	PA	✓	✓	✓	96.41	80.17

D Ablation

In the main paper, we have shown that our novel class-incremental learning strategy with neural meshes significantly outperforms the 2D baselines. In the following, we provide an analysis of how much the individual parts of our model contribute to this result.

Table 3 shows the contribution of each of our model components. We start with the most naive extension of NeMo to the class-incremental setting: in each task, we initialize the required number of meshes and fine-tune the feature extractor on each new task dataset. As expected, this leads to bad results. Next, we extend the models by the traditional distillation [5] (LwF) and herding exemplar [8] (iCaRL) strategies. The latter brings significant improvements. This shows that overall exemplar replay is necessary to retain knowledge while training Neural Mesh Models incrementally. We also compare applying LwF and iCaRL to either the 2D ResNet50 or NeMo and find that those strategies in most cases work better for the 2D setting, hence not being simply transferable to NeMo.

We finally demonstrate that our additions to maintain a structured latent space provide the best results by introducing the latent space initialization, positional regularization, and adding knowledge distillation. The results indicate that knowledge distillation has little effect (row 9), while adding the pose-aware replay (row 5 to row 6) has the largest impact on the result. This shows that the pose-aware exemplar selection strategy is critical and all other additions further improve the performance.

Table 4: Final task accuracy on Pascal3D with decreasing number of exemplars per class. Even with few exemplars, iNeMo retains good accuracy.

Metric	Exemplars	*P3D*
		$B0+3$
Classification	20	96.41
$\overline{acc}(1:N_{\text{task}})$ in % ↑	10	93.36
	5	82.59

E Training with less Replay Memory

Memory replay is essential when training iNeMo, as it allows updating neural meshes from previous tasks. However, storing too many samples per class in memory can become quite expensive and as such it is crucial for methods to be effective in utilizing replay with fewer samples. We show in Table 4 that iNeMo can adapt to lower memory sizes, but is optimal for the chosen 20 exemplars.

F Enhancing 2D Classifiers with Pose Annotations

Neural Mesh Models leverage meshes to host 3D consistent features and consequently, their training requires camera pose annotations. However, such pose annotations were not used in the 2D baselines, which could in principle give the Neural Mesh Models an advantage. To this end, we evaluate if using the pose annotation could improve the results of the 2D baselines. We extend the ResNet50 model with a second classifier head to predict the pose following [15] and use the following combined loss:

$$\mathcal{L}_{pe-cl} = \mathcal{L}_{iCaRL} + \lambda_{pe}\mathcal{L}_{Starmap}. \tag{1}$$

We then train the models in a class-incremental fashion with the iCaRL [8] strategy. The results are provided in Table 5 and show that the additional pose supervision introduced in this way does not help to improve the classification accuracy. When increasing the weight of the pose loss λ_{pe}, the performance consistently decreases with the best performing model being the default iCaRL network with $\lambda_{pe} = 0$.

G Additional Implementational Details

In this section, we provide the full implementation details about our method.

Data Preparation. NeMo [9] was originally proposed for 3D pose estimation, which means that the degrees of freedom to the camera pose are azimuth, elevation, and roll angle. This implies that the objects are scaled accordingly and

Table 5: Adding an additional pose estimation head and providing additional supervision does not lead to better representation learning. It is not obvious how conventional classifiers could leverage the additional camera pose annotation.

Metric	λ_{pe}	*P3D*
		$_{B0+3}$
	0.00	91.79
Classification	0.33	91.42
$\overline{acc}(1:N_{\text{task}})$ in % ↑	0.66	90.56
	1.00	89.86

centered in the images. We follow this procedure and use the publicly available code of NeMo. To make the sizes of all input images consistent, we further pad all images to the size of 640×800, where we fill the padded region with random textures from the Describable Textures Dataset [3].

Obtaining the 3D Cuboid Mesh is possible, since P3D [13] and O3D [12] provide a selection of 3D CAD models for each object category. For our 3D cuboid mesh, we consider the average bounding box of those models. We then sample vertices uniformly on its surface, leading to roughly 1,100 vertices per mesh.

Annotations at training time are computed with PyTorch3D's [7] mesh rasterizer. Concretely, we render the neural meshes with ground-truth camera poses to compute their projection and binary object masks. Additionally, we compute the projected coordinates $\pi(k)$ of each vertex V^k and its binary visibility o^k. Given the class label, we render the corresponding mesh at $\frac{1}{8}$ of the original image resolution (same size as the output of the feature extractor Φ). At each pixel, we determine vertex visibility by considering the closest face using the returned z-buffer. To parameterize the rasterizer, we use a relatively simple camera model with a focal length of 1. As there is no viewport specified for neither the P3D or OOD-CV [14] dataset, we follow previous work [4, 9, 11] and use a viewport of 3000/8. For the O3D [12] dataset we use their specified viewport of 2000/8.

H Pose Estimation

For the pose estimation, we follow previous work [4,9]. For completeness, we also provide a brief explanation here on how one can estimate the 3D object pose of an object of class c, given the trained feature extractor Φ and the neural mesh $\mathfrak{N}_c$.

3D Pose Estimation. During inference, we do not have access to the camera pose and corresponding perspective transformation. Since the camera intrinsics

and distance to the object are assumed to be known a-priori, we need to optimize for the unknown camera pose Q^{pred}. We define the foreground $\mathcal{F}$ in the same way as we did for the classification in Section 4.4 of the main part of the paper. However, in addition to pixels that have been recognized as background, we also remove pixel positions that fall outside the projection of the cuboid, leading to $\mathcal{F}_{\pi^{\text{pred}}} = \mathcal{F} \cap \{f_{\pi^{\text{pred}}(k)} | \forall V_k \in \mathcal{V}_c \wedge o_c^k = 1\}$.

Finding Q^{pred} is done via a render-and-compare approach. We do so by initializing a rough estimate and optimizing iteratively. Given the current camera pose and its incurred perspective transform π^{pred}, we maximize the current object likelihood:

$$\max_{Q^{\text{pred}}} \prod_{f_{\pi^{\text{pred}}(k)} \in \mathcal{F}_{\pi^{\text{pred}}}} P(f_{\pi^{\text{pred}}(k)} | \theta_c^k). \tag{2}$$

By considering the vMF distribution, we optimize the initial camera pose using PyTorch3D's [7] differentiable rasterizer by minimizing the negative log likelihood:

$$\mathcal{L}_{RC}(Q^{\text{pred}}) = \sum_{f_{\pi^{\text{pred}}(k)} \in \mathcal{F}_{\pi^{\text{pred}}}} -f_{\pi^{\text{pred}}(k)}^{\top} \cdot \theta_c^k. \tag{3}$$

Efficient Pose Estimation via Template Matching. The convergence of the above process is highly reliant on the provided initial pose, making it prohibitively slow in a worst case scenario. Wang *et al.* [10] proposed to speed it up by pre-rendering all neural meshes from 144 distinct viewing angles. Before the render-and-compare process, the output of the feature extractor is compared to each of these pre-rendered maps and the camera pose Q^{pred} is initialized with the pose that maximized the object likelihood. This simple procedure is remarkably effective, giving a speed-up of approximately 80% [4] over the original approach [9].

I Modelling the Background

For both classification and pose estimation, we leverage a set of features $\mathcal{B}$. This approach of disentangling foreground and background features into separate sets was introduced by Bai *et al.* [1]. Although we do not have a combined foreground set (but rather separate, 3D consistent meshes), we adopt their handling of the background model.

Learning the Background Model. We maintain a set $\mathcal{B}$ of N_{bg} features that are sampled from positions in the feature map that fall outside the cuboid projection. From each sample in a training batch of size b, we sample N_{bgupdate} new background features. Consequently, we need to remove $b \cdot N_{\text{bgupdate}}$ from $\mathcal{B}$ to avoid going over the allocated memory limit. Replacement is done, by maintaining a counter for each background feature, that indicates how many update steps it has been alive in $\mathcal{B}$ and prioritizing the oldest features for removal.

Balancing the Background Model. Ideally, the background should contain features from a wide variety of background options (*i.e.* water from boats, sky from airplanes, urban scenes from cars, ...). However, sampling background features from the current task dataset only means that $\mathcal{B}$ would be heavily biased towards background features from the currently considered classes. Therefore, we balance $\mathcal{B}$ after each training phase by sampling background features from the exemplar memory $\mathcal{E}^{1:i}$, which was constructed evenly from all classes and viewing angles.

J Hyperparameter Collection

As there are quite a few hyperparameters involved in our method, we include this brief section that notes down all parameters for our final model.

Opt.	LR	(β_1, β_2)	Task-Epoch	Batch Size	Weight Decay
Adam	1e-5	(0.9,0.999)	50	16	1e-4

Table 6: Optimization Parameters.

Table 7: Loss Weighting.

κ_1	κ_2	κ_3	λ_{etf}	λ_{kd}
1/0.07	1	0.5	0.1	10.0

Table 8: Mesh- and Background-related Parameters.

d	η	N_{bg}	N_{bgupdate}	R
128	0.9	2560	5	48

References

1. Bai, Y., Wang, A., Kortylewski, A., Yuille, A.: Coke: Localized contrastive learning for robust keypoint detection. Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (2023)

Table 9: Pose Estimation Parameters.

Opt.	LR	(β_1, β_2)	Epochs
Adam	5e-2	(0.4,0.6)	30

2. Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., Joulin, A.: Emerging properties in self-supervised vision transformers. In: Proceedings of the International Conference on Computer Vision (ICCV) (2021)
3. Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., , Vedaldi, A.: Describing textures in the wild. In: Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2014)
4. Jesslen, A., Zhang, G., Wang, A., Yuille, A., Kortylewski, A.: Robust 3d-aware object classification via discriminative render-and-compare. arXiv preprint arXiv:2305.14668 (2023)
5. Li, Z., Hoiem, D.: Learning without forgetting. TPAMI **40**(12), 2935–2947 (2018)
6. Liu, Y., Su, Y., Liu, A., Schiele, B., Sun, Q.: Mnemonics training: Multi-class incremental learning without forgetting. In: CVPR. pp. 12245–12254 (2020)
7. Ravi, N., Reizenstein, J., Novotny, D., Gordon, T., Lo, W.Y., Johnson, J., Gkioxari, G.: Accelerating 3d deep learning with pytorch3d. arXiv:2007.08501 (2020)
8. Rebuffi, S.A., Kolesnikov, A., Sperl, G., Lampert, C.H.: iCaRL: Incremental classifier and representation learning. In: CVPR. pp. 5533–5542 (2017)
9. Wang, A., Kortylewski, A., Yuille, A.: NeMo: Neural mesh models of contrastive features for robust 3d pose estimation. ICLR (2021)
10. Wang, A., Mei, S., Yuille, A.L., Kortylewski, A.: Neural view synthesis and matching for semi-supervised few-shot learning of 3d pose. NeurIPS **34**, 7207–7219 (2021)
11. Wang, A., Wang, P., Sun, J., Kortylewski, A., Yuille, A.: Voge: a differentiable volume renderer using gaussian ellipsoids for analysis-by-synthesis. In: ICLR (2022)
12. Xiang, Y., Kim, W., Chen, W., Ji, J., Choy, C., Su, H., Mottaghi, R., Guibas, L., Savarese, S.: Objectnet3d: A large scale database for 3d object recognition. In: ECCV (2016)
13. Xiang, Y., Mottaghi, R., Savarese, S.: Beyond pascal: A benchmark for 3d object detection in the wild. In: WACV. pp. 75–82. IEEE (2014)
14. Zhao, B., Yu, S., Ma, W., Yu, M., Mei, S., Wang, A., He, J., Yuille, A., Kortylewski, A.: Ood-cv: A benchmark for robustness to individual nuisances in real-world out-of-distribution shifts. In: ECCV (2022)
15. Zhou, X., Karpur, A., Luo, L., Huang, Q.: Starmap for category-agnostic keypoint and viewpoint estimation. In: ECCV. pp. 318–334 (2018)