HAPPI: Hyperbolic Hierarchical Prototypes for Image Recognition

Supplementary Material

1. Scaling Euclidean Features for Stable Hyperbolic Projection

In the MERU model [3], the extracted Euclidean features had an expected norm of $\sqrt{D}$ due to their CLIP-style layer initialization. This meant that when projected into hyperbolic space using the exponential map, their norm grew to approximately $e^{\sqrt{D}}$, which could cause numerical instability. To mitigate this, MERU applied a scaling strategy, introducing a learnable scalar α, which was initialized as $\frac{1}{\sqrt{D}}$. This ensured that feature norms remained controlled after projection, preventing overflow issues in hyperbolic space.

However, this initialization does not generalize across architectures. The norm of extracted features is not inherently $\sqrt{D}$; instead, it depends on various factors such as the backbone network, layer configurations, and activation functions. In our case, the Euclidean feature norms do not follow the same distribution as in MERU, making the fixed $\frac{1}{\sqrt{D}}$ initialization unsuitable. Rather than assuming a predefined norm, we empirically estimate it by computing the mean norm of features extracted from the first batch of training data. Specifically, let $\mathbb{E}[\|V_{\text{euc}}\|]$ denote the average norm of Euclidean feature vectors in this initial batch. We then initialize the learnable scalar α as:

$$\alpha = \frac{1}{\mathbb{E}[\|V_{\text{euc}}\|]} \tag{1}$$

This ensures that feature norms remain controlled when mapped to hyperbolic space, mitigating numerical instability.

Furthermore, this same scaling approach cannot be directly applied to prototype vectors. Since prototype vectors are learnable parameters independent of the feature extraction process, their norms do not necessarily align with those of extracted features. To maintain consistency, we explicitly scale the prototype vectors in Euclidean space so that their mean norm matches the estimated mean norm $\mathbb{E}[\|V_{\text{euc}}\|]$. That is, before projecting prototypes into hyperbolic space, we rescale them such that:

$$\mathbb{E}[\|P_{\text{euc}}\|] = \mathbb{E}[\|V_{\text{euc}}\|] \tag{2}$$

where P_{euc} represents the prototype vectors in Euclidean space.

By aligning the norm distributions of features and prototypes before projection, we ensure numerical stability while preserving a well-structured representation in hyperbolic space. This approach enables effective prototype-based classification without suffering from the norm explosion issues observed in prior work.

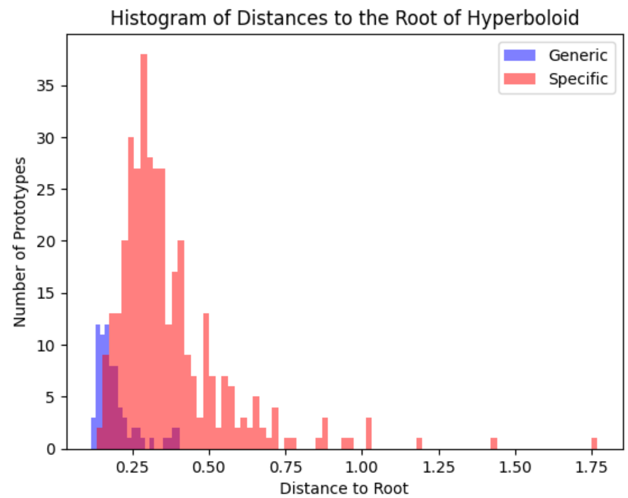

Figure 1. Distribution of distances from prototypes to the origin of the hyperboloid for generic and specific prototypes.

2. Placement of Prototypes in the Hyperbolic Space

To analyze the distribution of prototypes in hyperbolic space, we measured their distances from the origin of the hyperboloid. Figure 1 shows the distance distributions for generic and specific prototypes in HAPPI, using the XProtoNet [5] backbone, trained end-to-end (E2E) on the PETS dataset [7]. As illustrated, generic prototypes predominantly cluster closer to the origin, reflecting their role in capturing localized, distinctive features. This proximity aligns with our hierarchical organization where generic features are positioned near the origin of the hyperboloid. In contrast, specific prototypes are distributed farther from the root, indicating their role in aggregating broader patterns across larger regions.

3. Implementation Details

All models were trained using the original configurations presented in their respective papers unless stated otherwise. Below, we detail the specific training setup and modifications made for this study.

3.1. General Training Setup

For all models, PyTorch [8] was used for training, and Weights and Biases [1] was employed to log and monitor the training process. The experiments were conducted on NVIDIA Tesla V100 GPUs with 32GB of memory. Training was carried out for 100 epochs with the StepLR learning scheduler, which decays the learning rate by a factor of 0.8 every 5 epochs. Each model used 10 specific prototypes, and HAPPI-based models used 1 generic prototype, with one generic feature being extracted per input for HAPPI. The embedding depth D was set to 512 for all models, matching the depth of the extracted features and prototype vectors.

The optimizer used was Adam for most models, except for ProtoPFormer, where AdamW was used. The learning rate for all models was adjusted according to their original configurations, and all models used a batch size of 64. The training process also involved scaling methods to prevent numerical overflow during the exponential mapping of features to the hyperbolic space, which is further discussed in Section 1 of the supplementary material.

3.2. ProtoPNet

For ProtoPNet [2], the loss coefficients were set as follows: $\lambda_{\text{clstr\_g}} = 0.1$, $\lambda_{\text{sep\_g}} = 0.01$, $\lambda_{\text{clstr\_s}} = 0.8$, and $\lambda_{\text{sep\_s}} = 0.08$. The batch size was set to 64. The learning rates were configured as follows: for the backbone ResNet-50 [4] and the last layer fully connected classifier $h(.)$, a learning rate of 1×10^{-4} was used, while for the rest of the model, a learning rate of 3×10^{-3} was applied. When using HAPPI, the learning rate for the curvature of the hyperbolic space and the scaling factor α was set to 5×10^{-4}. To train the end-to-end (E2E) version, for both Euclidean and HAPPI versions, we used a uniform learning rate of 1×10^{-4} for all parameters.

3.3. XProtoNet

For XProtoNet, the loss coefficients were the same as ProtoPNet: $\lambda_{\text{clstr\_g}} = 0.1$, $\lambda_{\text{sep\_g}} = 0.01$, $\lambda_{\text{clstr\_s}} = 0.8$, and $\lambda_{\text{sep\_s}} = 0.08$. The batch size was 36 with gradient accumulation steps of 2. The learning rates for the original version were set as follows: for the ResNet-50 backbone and the last layer fully connected classifier $h(.)$, a learning rate of 1×10^{-4} was used, and for the rest of the model, the learning rate was 3×10^{-3}. In the HAPPI version, the learning rate for the curvature of the hyperbolic space and the scaling factor α was set to 5×10^{-4}. The end-to-end (E2E) version used a uniform learning rate of 1×10^{-4} for all parameters.

3.4. MCPNet

For MCPNet [9], we used their published code repositories and reproduced their method without using the center-crop functionality for the images, as used in their original repository.

3.5. PipNet

For PipNet [6], we used the same configurations as those presented in their original paper.

3.6. ST-ProtoPNet

For ST-ProtoPNet [10], the batch size was set to 64, in line with the original paper's configuration.

3.7. ProtoPFormer

For ProtoPFormer [11], the batch size was set to 64, and we used the AdamW optimizer as specified in the original paper. Instead of the Prototypical Part Concentration (PPC) loss, we implemented our clustering and separation loss functions to better align prototypes in hyperbolic space. The CLS token was used as the generic prototype, while the image tokens were treated as specific prototypes.

3.8. Black-Box Baselines

For the black-box baseline, the batch size was set to 64, in line with the configurations used for other models.

References

[1] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com. 1

[2] Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan K Su. This looks like that: deep learning for interpretable image recognition. *Advances in neural information processing systems*, 32, 2019. 2

[3] Karan Desai, Maximilian Nickel, Tanmay Rajpurohit, Justin Johnson, and Ramakrishna Vedantam. Hyperbolic image-text representations. *arXiv (Cornell University)*, 2023. 1

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2

[5] Eunji Kim, Siwon Kim, Minji Seo, and Sungroh Yoon. Xprotonet: diagnosis in chest radiography with global and local explanations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15719–15728, 2021. 1

[6] Meike Nauta, Jörg Schlötterer, Maurice Van Keulen, and Christin Seifert. Pip-net: Patch-based intuitive prototypes for interpretable image classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2744–2753, 2023. 2

[7] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. Cats and dogs. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3498–3505. IEEE, 2012. 1

[8] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017. 1

[9] Bor-Shiun Wang, Chien-Yi Wang, and Wei-Chen Chiu. Mcpnet: An interpretable classifier via multi-level concept prototypes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10885–10894, 2024. 2

[10] Chong Wang, Yuyuan Liu, Yuanhong Chen, Fengbei Liu, Yu Tian, Davis McCarthy, Helen Frazer, and Gustavo Carneiro. Learning support and trivial prototypes for interpretable image classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2062–2072, 2023. 2

[11] Mengqi Xue, Qihan Huang, Haofei Zhang, Lechao Cheng, Jie Song, Minghui Wu, and Mingli Song. Protopformer: Concentrating on prototypical parts in vision transformers for interpretable image recognition. *arXiv preprint arXiv:2208.10431*, 2022. 2