Exploiting Interpretable Capabilities with Concept-Enhanced Diffusion and Prototype Networks

Alba Carballo-Castro, Sonia Laguna, Moritz Vandenhirtz, Julia E. Vogt
Department of Computer Science
ETH Zürich
Switzerland

Abstract

Concept-based machine learning methods have increasingly gained importance due to the growing interest in making neural networks interpretable. However, concept annotations are generally challenging to obtain, making it crucial to leverage all their prior knowledge. By creating concept-enriched models that incorporate concept information into existing architectures, we exploit their interpretable capabilities to the fullest extent. In particular, we propose Concept-Guided Conditional Diffusion, which can generate visual representations of concepts, and Concept-Guided Prototype Networks, which can create a concept prototype dataset and leverage it to perform interpretable concept prediction. These results open up new lines of research by exploiting pre-existing information in the quest for rendering machine learning more human-understandable.

1 Introduction

With an increasing number of decision-making processes relying on Machine Learning (ML) methods, the field of Interpretable ML has gained significant importance (Doshi-Velez & Kim, 2017; Lipton, 2016) with concept-based methods being a prominent focus of the recent literature in this area. Concepts can be defined as variables that encode human-understandable information and can be used to provide explanations. Concept-based methods have been explored in prior work (Lampert et al., 2009; Kumar et al., 2009), with Concept Bottleneck Models (CBMs) (Koh et al., 2020), Concept Embedding Models (CEMs) (Espinosa Zarlenga et al., 2022) Concept Activation Vectors (CAVs) (Kim et al., 2018), and Concept Whitening (Z. Chen et al., 2020) among the most popular.

In this work, we explore how concept information can help increase interpretability beyond its current use by integrating it into pre-existing methods. We propose different means to leverage concept knowledge and obtain visual concept representations by introducing *Concept-Guided Conditional Diffusion* and *Concept-Guided Prototype Networks*. The former is based on diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020), particularly on conditional diffusion (Ho & Salimans, 2021), to use the concept information for guidance and obtain visual representations of the concepts. The latter builds on already interpretable methods such as Prototypical Part Networks (C. Chen et al., 2019) to obtain prototypical patches (prototypes). They characterize concept information and allow for concept prediction and creating a *concept prototype dataset*.

Contributions This work contributes to the line of research on concept-based methods in several ways by introducing concept-enhanced methods to leverage concept knowledge and exploit their interpretable capabilities. (*i*) We present *Concept-Guided Conditional Diffusion*, a generative method that incorporates concept knowledge to guide the generation of concept-based samples. (*ii*) We introduce *Concept-Guided Prototype Networks*, which allow us to (a) obtain interpretable concept predictions and (b) create a concept prototype dataset including a visual representation of the concepts. (*iii*) We illustrate the success of these methods through real-world dataset applications.

Workshop on Scientific Methods for Understanding Deep Learning, NeurIPS 2024.

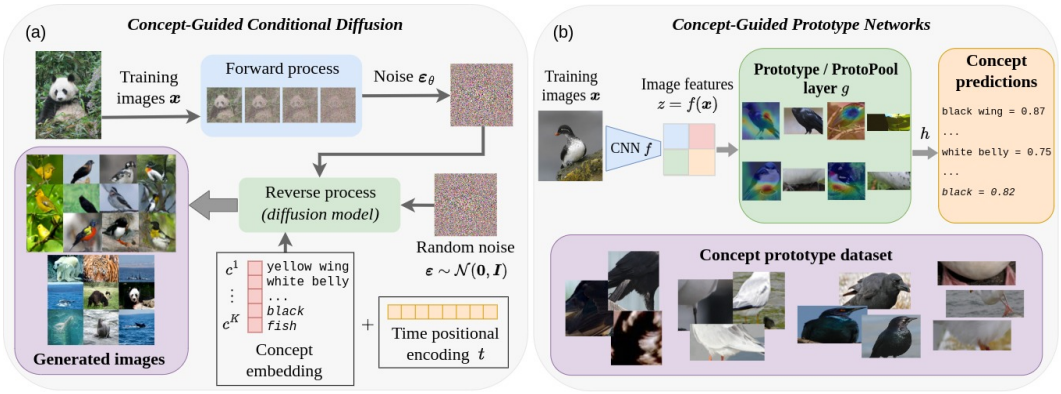

Figure 1: Summary of the methodology of (a) Concept-Guided Conditional Diffusion and (b) Prototype Networks. The resulting generations and prototype visualizations are shown in purple.

2 Related Work

Concept-based methods Initially, concept-based methods aimed at attribute based classification (Lampert et al., 2009; Kumar et al., 2009). Later on, Concept Activation Vectors (TCAV) (Kim et al., 2018) and Invertible Concept-based Explanations (ICE) (Zhang et al., 2021) were proposed as methods to interpret neural networks through human-understandable concepts. Koh et al. (2020) proposed Concept Bottleneck Models (CBM), which enforce an intermediate layer to correspond to concepts. Derivative works find ways to relax the bottleneck (Espinosa Zarlenga et al., 2022), don't require concept information during training (Oikarinen et al., 2023; Yuksekgonul et al., 2023; Laguna et al., 2024), or improve the modeling capabilities (Sawada & Nakamura, 2022; Havasi et al., 2022; Vandenhirtz et al., 2024). More recent works consider the concept bottleneck in the context of generative methods (Marconato et al., 2022; Ismail et al., 2024). Contrary to the restrictive enforcing of a concept bottleneck, our proposed generative method conditions on the complete concept information, allowing steerable generations of not only positive concept information (i.e., images containing a concept) but also negative, exploiting concept knowledge more exhaustively.

Diffusion models Diffusion models were originally introduced by Sohl-Dickstein et al. (2015), and have been followed by a series of works focusing on image generation and improved performance, such as noise conditional score network (NCSN) (Y. Song & Ermon, 2019), Denoising Diffusion Probabilistic Models (DDPM) (Ho et al., 2020) or Denoising Implicit Diffusion Models (DDIM) (J. Song et al., 2022). Further works have dealt with Conditional Diffusion using a classifier as guidance (Dhariwal & Nichol, 2021) or without making use of it (Ho & Salimans, 2021).

Prototype networks They perform classification by computing the similarity of the encoded input with a prototypical embedding (Li et al., 2018), which can be restricted to match an actual image patch (C. Chen et al., 2019). Other related works are ProtoPShare (Rymarczyk et al., 2021), ProtoTrees (Nauta et al., 2021), ProtoPool (Rymarczyk et al., 2022) and ProtoConcepts (Ma et al., 2023), that learn prototypical concepts from the data rather than using a pre-annotated concept set.

3 Methods

Concept-based methods are trained on triplets $\{(\boldsymbol{x}_i, y_i, \boldsymbol{c}_i\}_{i=1}^n$, where $\boldsymbol{x} \in \mathbb{R}^d$ are the inputs, $y \in \mathbb{R}$ is the target, and $\boldsymbol{c} \in \{0, 1\}^k$ are different human-understandable concepts. We propose to incorporate the K concepts $\boldsymbol{c} = c^1, \dots c^K$ into pre-existing methods to create concept-enhanced models[1]. This way, we acquire visual representations of concepts to enhance human interpretability. A summary of our methods is illustrated in Figure 1. In the following, we explore two ways to infuse pre-existing methods with concept information to obtain visual representations.

[1]The code is publicly available here: `https://github.com/acarballocastro/ConceptEnhanced`

3.1 Concept-Guided Conditional Diffusion

Previous works on Conditional Diffusion use label information to guide the generations, but focus mostly on the multi-class setting (Dhariwal & Nichol, 2021; Ho & Salimans, 2021). Our method extends classifier-free guidance (Ho & Salimans, 2021) to the multi-binary-label case to leverage concept knowledge, using the concept vector $\boldsymbol{c}$ to guide the diffusion process. The original approach passes label information via a learnable embedding that is added to the timestep positional encoding. Our method replaces the label information with a *concept embedding* with K rows, one per concept.

Each datapoint $\boldsymbol{x}$ has per-concept binary annotations, representing its presence in the image. To guide the image generation with a subset of concepts, we introduce a user-defined binary mask $\boldsymbol{m}$, where $m^k = 1$ if the concept is activated. Given an embedding $\mathcal{E} = (\boldsymbol{e}_k)_{k=1}^K$, we examine three ways of selecting the rows of the mask-activated concepts (see Figure 3 in Appendix A.1 for more details): **Positive**, where the $\boldsymbol{e}_k$ are selected if $m_i^k = 1$ and $c_i^k = 1$; **Opposite**, which selects all rows activated by the mask $m_i^k = 1$ but inverts, $-\boldsymbol{e}_k$, if $c_i^k = 0$; and **Double**, which initializes two embeddings $\mathcal{E}_1$ and $\mathcal{E}_2$ and extracts the rows where $m_i^k = 1$ from $\mathcal{E}_1$ when $c_i^k = 1$, and $\mathcal{E}_2$ when $c_i^k = 0$.

Finally, the extracted rows are averaged and added to the positional encoding of the timesteps, used to guide the generations in the conditional model. Our proposed modelling allows us to introduce the guidance of generations not only with positive concepts $c_i^k = 1$ but also with negative concepts $c_i^k = 0$ (i.e., generations in which the concept of interest is *not* present).

3.2 Concept-Guided Prototype Networks

In this section, we introduce the extension of ProtoPNet (C. Chen et al., 2019) and its subsequent work ProtoPools (Rymarczyk et al., 2022) to concept-enhanced methods by adapting them from the multi-class to the multi-binary label setting, allowing for interpretable concept prediction. In addition, prototypical image patches (prototypes) will be obtained for positive ($c_i^k = 1$) and negative ($c_i^k = 0$) concepts, resulting in $2 \times K$ concept classes for prototype training but just K for concept prediction.

Concept-Guided ProtoPNet Given an input image $\boldsymbol{x}$, ProtoPNet (C. Chen et al., 2019) is composed by a CNN layer f that extracts image features $\boldsymbol{z} = f(\boldsymbol{x})$, a prototype layer $g_{\boldsymbol{p}}$ to compute similarities between prototypes and patches of the convolutional output, and a fully connected layer h for the final classification task. A summary of the original training algorithm is available in Appendix A.2.

If $\boldsymbol{P} = \{\boldsymbol{p}_j\}_{j=1}^m$ are the different m prototypes, Concept-Guided ProtoPNet modifies the loss function in Equation 4 by replacing the *cross entropy* for the *binary cross entropy* and generalize the cluster (Clst) and separation (Sep) costs through the distance loss

$$\text{Dist} = \frac{1}{n}\frac{1}{K}\sum_{i=1}^{n}\sum_{k=1}^{K}\min_{j:\boldsymbol{p}_j\in\mathcal{P}}\min_{\boldsymbol{z}\in\text{patches}(f(x_i))}\|\boldsymbol{z}-\boldsymbol{p}_j\|_2^2, \tag{1}$$

We partition $\boldsymbol{P} = \boldsymbol{P}_k^{c_i^k} \cup \boldsymbol{P}_k^{\bar{c}_i^k}$ into the subsets of prototypes assigned to the concept classes of x_i and those assigned to the opposite, respectively. Clst will be Dist calculated for $\mathcal{P} = \boldsymbol{P}_k^{c_i^k}$, encouraging to have at least one latent patch in every training image close to at least one prototype of its concept class for the different K concepts. Sep is calculated as $-$Dist for $\mathcal{P} = \boldsymbol{P}_k^{\bar{c}_i^k}$, where c_i^k is the binary value of the k-th concept for the i-th training sample and $\bar{c}_i^k = 1 - c_i^k$. For each concept, it favours that there are no latent patches close to the prototypes assigned to the opposite concept classes.

For the prototype projection step, the update remains unchanged while accounting for the subsets of patches per concept being positive and negative. Finally, as the concepts are now not mutually exclusive, the weights of the last layer are initialized as $w_h^{(k,j)} = 1$ if $\boldsymbol{p}_j \in \boldsymbol{P}_k^1$, and $w_h^{(l,j)} = -1$ if $\boldsymbol{p}_j \notin \boldsymbol{P}_k^0$ for a given concept k. That is, the weights linking a positive concept logit with its prototypes are set to 1, whereas the ones linking the negative concept logits to its (negative) prototypes are set to -1. The rest of the weights are initialized to 0, encouraging a better classification.

Concept-Guided ProtoPools In ProtoPools (Rymarczyk et al., 2022), prototypes can be shared across classes, and the allocation of prototypes to classes is dynamical instead of predetermined. The

model uses prototype pool layer g, which learns a set of m prototypes $\boldsymbol{P} = \{\boldsymbol{p}_j\}_{j=1}^{m}$ and a set of S distributions $q_s \in \mathbb{R}^m$ per class. Each class has S *slots*, and the distributions represent the probability of a prototype being assigned to one of the slots of that class. Layer g now computes S similarity scores per class which then pass by the fully connected layer to output the final prediction.

The main differences in the training algorithm with respect to ProtoPNet are the dynamical assignment of prototypes using a Gumbel-Softmax distribution and a new orthogonal loss to ensure that the same prototype is not assigned to more than one slot per class. The projection of prototypes and convex optimization of the last layer steps are also adapted (see Appendix A.2 for further details).

To introduce Concept-Guided ProtoPools, we adapt it to the multi-binary-label case by calculating the cluster and separation costs as described before. In addition to the orthogonal loss in Equation 10, a second orthogonal loss is introduced to ensure the same prototype is not assigned to the positive and negative classes of a given concept:

$$\text{Orth}_c = \sum_{i,j}^{S} \frac{\langle q_i, q_j \rangle}{\|q_i\|_2 \cdot \|q_j\|_2}, \tag{2}$$

where $i = 1, \ldots, S$ are the indexes of q_i the distributions of concept c^k and $j = 1, \ldots, S$ are the indexes of q_j the distributions of the opposite concept $\bar{c}^k$, with $k = 1, \ldots, K$.

The final loss function has the form:

$$\mathcal{L}_{CGPPool} = \frac{1}{n} \sum_{i=1}^{n} BCE\big[(h \circ g_{\boldsymbol{p}} \circ f)(x_i),\ y_i\big] + \lambda_1 \text{Clst} + \lambda_2 \text{Sep} + \lambda_3 \text{Orth}_p + \lambda_4 \text{Orth}_c. \tag{3}$$

For the projection of prototypes, $\mathcal{Z}_j$ stays the same as in Equation 11. Prototypes are assigned to the different S slots of the $2 \times K$ concept classes, and then the model pushes each of them to the nearest training patch. Finally, for the convex optimization of the last layer, we set to 1 the weights that link a given positive concept to its S slots and to -1 those that link a negative concept logit and its slots.

4 Results

Experiments were performed on the Caltech-UCSD Birds200-2011 dataset (Wah et al., 2011), in its adaptation from the CBM literature (Koh et al., 2020) with 112 concepts and 200 classes, and the Animals with Attributes 2 (AWA2) dataset (Xian et al., 2019) comprising 85 concepts and 50 classes. We refer to Appendix B for more details on the experimental setup.

Concept-Guided Conditional Diffusion Figure 2(a) shows example generations for different concepts, complemented by the sample images in Figure 1 and a comprehensive overview in Appendix C.1 (for CUB) and Appendix C.2 (for AWA2). The concepts of interest are present in all generations and, therefore, the model has been capable of generating visual concept representations with no remarkable differences between embedding types. In addition, generations also work for negative concepts, as it can be seen that the resulting images do not contain the concept when it is set to negative. In addition, generations for concept `black wing` are better defined and have better quality than those of `yellow wing`, which is due to a higher proportion of images containing this concept. For guidance with multiple concepts at once, the desired generations for both positive and negative concepts are obtained. For instance, generations for concept `fish` positive and `black` negative result in images of animals that have fish in their diet but are not of color black. With this, our method exploits the expressivity of concepts, particularly when a direct human interpretation is not trivial.

Concept-Guided Prototype Networks Figure 2(b) shows generated prototypes for different concepts. The model can identify and extract relevant patches of Figures 18, 19, 20 and 21 in Appendix D display additional examples of prototypes alongside their closest patches for positive and negative concepts. Accuracy results for the different base architectures are shown in Tables 3 (for ProtoPNet) and 4 (for ProtoPools) in Appendix D. Despite a slight drop in concept accuracy with respect to the oracle, concept prototypes provide improved interpretability through a better understanding of the way models visualize concepts. In particular, AwA2 behaves on par with the oracle performance, especially in Concept-Guided ProtoPools, which highlights the method's success in bigger datasets. These methods allow to leverage concept information which can be used to increase downstream interpretability in other ML models.

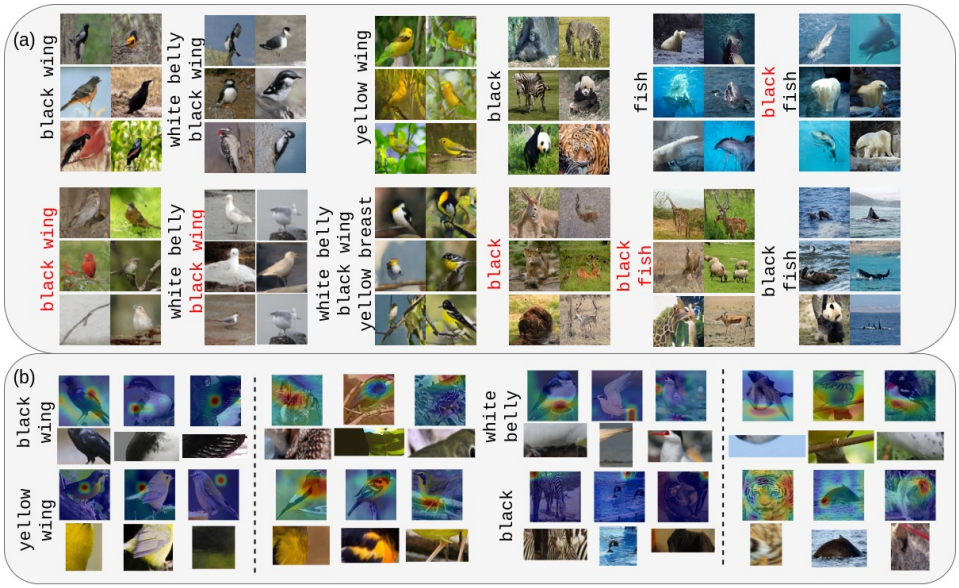

Figure 2: (a) Concept generations for `positive` and `negative` concept guidance. Each row corresponds to a different embedding type (positive, opposite, double). (b) Extracted concept prototypes and their activation maps for CG-ProtoPNet (left) and CG-ProtoPools (right).

5 Conclusion

In this paper, we introduced concept-enhanced methods as a way to incorporate concept information into existing architectures, and thus exploit their interpretable capabilities to the fullest extent. We proposed two different models *Concept-Guided Conditional Diffusion* and *Concept-Guided Prototype Networks*, which can generate concept-based samples or *prototypes* and perform interpretable concept prediction. We applied these models to real-world datasets and showed successful image generation capabilities. This allows the visualization of concept information and an interpretable concept prediction without significant loss of performance with respect to black-box methods.

Limitations and future work This work can be explored and impactful in the continuous concept domain. Another application lies in the context of CBMs, which are known to suffer from data leakage (Mahinpei et al., 2021; Margeloiu et al., 2021; Havasi et al., 2022), where the concept predictor conveys unintended information about the label. This issue could be further understood and mitigated with concept-enhanced methods, such as substituting usual black-box predictors for an interpretable one as a Concept-Guided Prototype Network; or calculating concept probabilities by similarity to concept prototypes or samples generated with Concept-Guided Conditional Diffusion. Lastly, both Concept-Guided methods could prove useful in obtaining visualizations to better understand the side channel. As a limitation, our method relies on concept-annotated datasets, which can be overcome in combination with current works on automated concept discovery (Ghorbani et al., 2019; Oikarinen et al., 2023). Finally, generating images with diffusion models is computationally expensive and could be alleviated with recent less demanding variations (Rombach et al., 2021; Guo et al., 2024).

Acknowledgments and Disclosure of Funding

The project that gave rise to these results received the support of a fellowship from "la Caixa" Foundation (ID 100010434). The fellowship code is LCF/BQ/EU22/11930089. MV and SL are supported by the Swiss State Secretariat for Education, Research, and Innovation (SERI) under contract number MB22.00047.

References

Chen, C., Li, O., Tao, C., Barnett, A. J., Su, J., & Rudin, C. (2019). This looks like that: Deep learning for interpretable image recognition. In *33rd conference on neural information processing systems.*

Chen, Z., Bei, Y., & Rudin, C. (2020). Concept whitening for interpretable image recognition. *Nature Machine Intelligence*, *2*(12), 772–782. Retrieved from https://doi.org/10.1038/s42256-020-00265-z

Dhariwal, P., & Nichol, A. (2021). Diffusion models beat gans on image synthesis. In *Advances in neural information processing systems* (Vol. 34, pp. 8780–8794).

Doshi-Velez, F., & Kim, B. (2017, March). *Towards A Rigorous Science of Interpretable Machine Learning* (No. arXiv:1702.08608). arXiv. doi: 10.48550/arXiv.1702.08608

Espinosa Zarlenga, M., Barbiero, P., Ciravegna, G., Marra, G., Giannini, F., Diligenti, M., ... others (2022). Concept embedding models: Beyond the accuracy-explainability trade-off. In *Advances in neural information processing systems* (Vol. 35, pp. 21400–21413).

Ghorbani, A., Wexler, J., Zou, J. Y., & Kim, B. (2019). Towards automatic concept-based explanations. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32, p. 9277—9286). Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2019/file/77d2afcb31f6493e350fca61764efb9a-Paper.pdf

Guo, L., He, Y., Chen, H., Xia, M., Cun, X., Wang, Y., ... Wen, B. (2024). Make a cheap scaling: A self-cascade diffusion model for higher-resolution adaptation. *arXiv preprint arxiv:2402.10491*. Retrieved from https://arxiv.org/abs/2402.10491

Havasi, M., Parbhoo, S., & Doshi-Velez, F. (2022). Addressing leakage in concept bottleneck models. In A. H. Oh, A. Agarwal, D. Belgrave, & K. Cho (Eds.), *Advances in neural information processing systems.* Retrieved from https://openreview.net/forum?id=tglniD_fn9

Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. *arXiv preprint arxiv:2006.11239*.

Ho, J., & Salimans, T. (2021). Classifier-free diffusion guidance. In *Neurips 2021 workshop on deep generative models and downstream applications.*

Ismail, A. A., Adebayo, J., Bravo, H. C., Ra, S., & Cho, K. (2024). Concept bottleneck generative models. In *The twelfth international conference on learning representations.* Retrieved from https://openreview.net/forum?id=L9U5MJJleF

Jang, E., Gu, S., & Poole, B. (2017). *Categorical reparameterization with gumbel-softmax.* Retrieved from https://arxiv.org/abs/1611.01144

Kim, B., Wattenberg, M., Gilmer, J., Cai, C., Wexler, J., Viegas, F., & sayres, R. (2018). Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In J. Dy & A. Krause (Eds.), *Proceedings of the 35th international conference on machine learning* (Vol. 80, p. 2668-2677). PMLR. Retrieved from http://proceedings.mlr.press/v80/kim18d.html

Koh, P. W., Nguyen, T., Tang, Y. S., Mussmann, S., Pierson, E., Kim, B., & Liang, P. (2020). Concept bottleneck models. In H. D. III & A. Singh (Eds.), *Proceedings of the 37th international conference on machine learning* (Vol. 119, pp. 5338–5348). Virtual: PMLR. Retrieved from https://proceedings.mlr.press/v119/koh20a.html

Kumar, N., Berg, A. C., Belhumeur, P. N., & Nayar, S. K. (2009). Attribute and simile classifiers for face verification. In *2009 ieee 12th international conference on computer vision* (pp. 365–372). Kyoto, Japan: IEEE. Retrieved from https://doi.org/10.1109/ICCV.2009.5459250

Laguna, S., Marcinkevičs, R., Vandenhirtz, M., & Vogt, J. E. (2024). Beyond concept bottleneck models: How to make black boxes intervenable? *arXiv preprint arXiv:2401.13544*.

Lampert, C. H., Nickisch, H., & Harmeling, S. (2009). Learning to detect unseen object classes by between-class attribute transfer. In *2009 IEEE conference on computer vision and pattern recognition.* Miami, FL, USA: IEEE. Retrieved from https://doi.org/10.1109/CVPR.2009.5206594

Li, O., Liu, H., Chen, C., & Rudin, C. (2018). Deep learning for case-based reasoning through prototypes: a neural network that explains its predictions. In *Proceedings of the thirty-second aaai conference on artificial intelligence and thirtieth innovative applications of artificial intelligence conference and eighth aaai symposium on educational advances in artificial intelligence.* AAAI Press.

Lipton, Z. C. (2016, June). The Mythos of Model Interpretability. *Communications of the ACM*, *61*(10), 35–43. doi: 10.48550/arxiv.1606.03490

Ma, C., Zhao, B., Chen, C., & Rudin, C. (2023). This looks like those: Illuminating prototypical concepts using multiple visualizations. In *Thirty-seventh conference on neural information processing systems.* Retrieved from https://openreview.net/forum?id=dCAk9VlegR

Maddison, C. J., Mnih, A., & Teh, Y. W. (2017). *The concrete distribution: A continuous relaxation of discrete random variables.* Retrieved from https://arxiv.org/abs/1611.00712

Mahinpei, A., Clark, J., Lage, I., Doshi-Velez, F., & Pan, W. (2021). *Promises and pitfalls of black-box concept learning models.* Retrieved from https://doi.org/10.48550/arXiv.2106.13314 (*arXiv:2106.13314*)

Marconato, E., Passerini, A., & Teso, S. (2022). *GlanceNets: Interpretable, leak-proof concept-based models.* (*arXiv:2205.15612*)

Margeloiu, A., Ashman, M., Bhatt, U., Chen, Y., Jamnik, M., & Weller, A. (2021). *Do concept bottleneck models learn as intended?* Retrieved from https://doi.org/10.48550/arXiv.2105.04289 (*arXiv:2105.04289*)

Nauta, M., van Bree, R., & Seifert, C. (2021). *Neural prototype trees for interpretable fine-grained image recognition.* Retrieved from https://arxiv.org/abs/2012.02046

Oikarinen, T., Das, S., Nguyen, L. M., & Weng, T.-W. (2023). Label-free concept bottleneck models. In *The eleventh international conference on learning representations.* Retrieved from https://openreview.net/forum?id=FlCg47MNvBA

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2021). *High-resolution image synthesis with latent diffusion models.*

Rymarczyk, D., Struski, Ł., Górszczak, M., Lewandowska, K., Tabor, J., & Zieliński, B. (2022). Interpretable image classification with differentiable prototypes assignment. In *European conference on computer vision* (pp. 351–368).

Rymarczyk, D., Struski, Ł., Tabor, J., & Zieliński, B. (2021, August). Protopshare: Prototypical parts sharing for similarity discovery in interpretable image classification. In *Proceedings of the 27th acm sigkdd conference on knowledge discovery and data mining.* ACM. Retrieved from http://dx.doi.org/10.1145/3447548.3467245 doi: 10.1145/3447548.3467245

Sawada, Y., & Nakamura, K. (2022). Concept bottleneck model with additional unsupervised concepts. *IEEE Access*, *10*, 41758–41765. Retrieved from https://doi.org/10.1109/ACCESS.2022.3167702

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., & Ganguli, S. (2015, 07–09 Jul). Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of the 32nd international conference on machine learning* (Vol. 37, pp. 2256–2265). PMLR.

Song, J., Meng, C., & Ermon, S. (2022). Denoising diffusion implicit models. *arXiv preprint arxiv:2010.02502*.

Song, Y., & Ermon, S. (2019). Generative modeling by estimating gradients of the data distribution. In *Advances in neural information processing systems* (pp. 11895–11907).

Vandenhirtz, M., Laguna, S., Marcinkevičs, R., & Vogt, J. E. (2024). Stochastic concept bottleneck models. In *Icml 2024 workshop on structured probabilistic inference & generative modeling*. Retrieved from `https://openreview.net/forum?id=8jG3Y0xX7b`

Wah, C., Branson, S., Welinder, P., Perona, P., & Belongie, S. (2011). The caltech-ucsd birds-200-2011 dataset.

Xian, Y., Lampert, C. H., Schiele, B., & Akata, Z. (2019, sep). Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, *41*(09), 2251-2265. doi: 10.1109/TPAMI.2018.2857768

Yuksekgonul, M., Wang, M., & Zou, J. (2023). Post-hoc concept bottleneck models. In *The 11th international conference on learning representations*. Retrieved from `https://openreview.net/forum?id=nA5AZ8CEyow`

Zhang, R., Madumal, P., Miller, T., Ehinger, K. A., & Rubinstein, B. I. P. (2021, May). Invertible concept-based explanations for cnn models with non-negative concept activation vectors. *Proceedings of the AAAI Conference on Artificial Intelligence*, *35*(13), 11682-11690. Retrieved from `https://ojs.aaai.org/index.php/AAAI/article/view/17389` doi: 10.1609/aaai.v35i13.17389

A Methods

In this section, we provide the reader with further details for a better understanding of the concept-enhanced methods implemented.

A.1 Concept-Guided Conditional Diffusion

Figure 3 summarizes the three types of embedding for Concept-Guided Conditional Diffusion. In the *positive embedding*, the situation in which a concept is not present for an image is treated similarly to when the concept is not relevant to guide the generations. In the *opposite embedding*, the idea is inspired by methods from Natural Language Processing, where it is usual to force an embedding dimension to encode a semantic quality. Finally, for the *double embedding*, it is ensured that positive and negative concepts are learned independently.

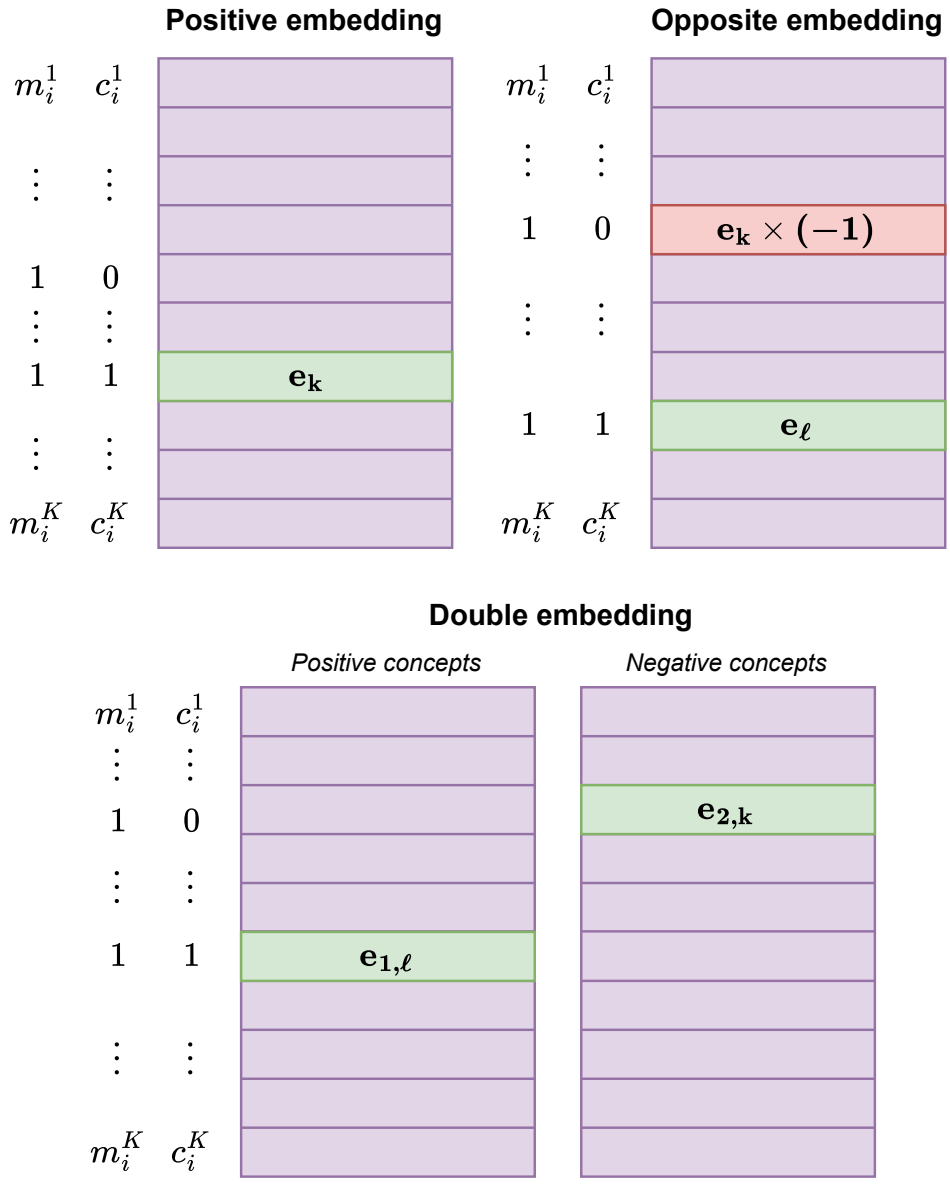

Figure 3: Schematic representation of the three types of embedding proposed for the Concept-Guided Conditional Diffusion model.

A.2 Concept-Guided Prototype Networks

The steps to train a Prototypical Part Network (ProtoPNet) as originally described in C. Chen et al. (2019) are the following:

(1) SDG of layers before the last layer In this step, the model should learn the most representative patches of each class for the later classification of the images (i.e. the prototypes). The optimization problem is therefore posed on the convolutional layer parameters w_{conv} and the different prototypes $\boldsymbol{P} = \{\boldsymbol{p}_j\}_{j=1}^m$ of $g_{\boldsymbol{p}}$. The aim is to minimize the loss:

$$\mathcal{L}_{PPNet} = \frac{1}{n}\sum_{i=1}^{n} CrossEntropy\big[(h \circ g_{\boldsymbol{p}} \circ f)(x_i),\ y_i\big] + \lambda_1 \text{Clst} + \lambda_2 \text{Sep}, \tag{4}$$

where

$$\text{Clst} = \frac{1}{n}\sum_{i=1}^{n} \min_{j:\boldsymbol{p}_j \in \boldsymbol{P}_{y_i}} \min_{\boldsymbol{z} \in \text{patches}(f(x_i))} \|\boldsymbol{z} - \boldsymbol{p}_j\|_2^2, \tag{5}$$

and

$$\text{Sep} = -\frac{1}{n}\sum_{i=1}^{n} \min_{j:\boldsymbol{p}_j \notin \boldsymbol{P}_{y_i}} \min_{\boldsymbol{z} \in \text{patches}(f(x_i))} \|\boldsymbol{z} - \boldsymbol{p}_j\|_2^2. \tag{6}$$

The first term of the loss, the cross entropy, penalizes incorrect classification of the training images. The cluster cost (Clst) encourages that there is at least one latent patch in every training image that is close to at least one prototype of its own class. Finally, the separation cost (Sep) encourages that all patches from a training image stay far from all the prototypes that are not of the same class.

(2) Projection of prototypes To equate each of the prototypes that are in the latent space to actual patches of the training images, the model *pushes* each prototype $\boldsymbol{p}_j$ to the nearest latent patch from the same class with the update

$$\boldsymbol{p}_j \leftarrow \arg\min_{\boldsymbol{z} \in \mathcal{Z}_j} \|\boldsymbol{z} - \boldsymbol{p}_j\|_2, \quad \mathcal{Z}_j = \{\tilde{\boldsymbol{z}} : \tilde{\boldsymbol{z}} \in \text{patches}(f(\boldsymbol{x}_i)), \forall i : y_i = l\}, \tag{7}$$

where $l = 1, \ldots, L$ are the different L classes or labels.

(3) Convex optimization of the last layer The weights of the last layer between the prototypes and the class logits are initialized as $w_h^{(l,j)} = 1$ if $\boldsymbol{p}_j \in \boldsymbol{P}_l$ and as $w_h^{(l,j)} = -0.5$ if $\boldsymbol{p}_j \notin \boldsymbol{P}_l$, where $\boldsymbol{P}_l \subseteq \boldsymbol{P}$ are the prototypes of class l. That is, depending on whether the weight connects the prototype of a given class with its class logit or not.

The authors propose a training step in which adjust the last layer connection weights $w_h^{(l,j)}$ to ensure sparsity by making $w_h^{(l,j)} \approx 0$ when $\boldsymbol{p}_j \notin \boldsymbol{P}_l$. According to the authors, this ensures that the model relies less on a negative reasoning process. Fixing all the parameters in the previous convolutional and prototype layers, the resulting convex optimization problem is

$$\min_{w_h} \frac{1}{n}\sum_{i=1}^{n} CrossEntropy\big[(h \circ g_{\boldsymbol{p}} \circ f)(x_i),\ y_i\big] + \lambda_{last} \sum_{l=1}^{L} \sum_{j:\boldsymbol{p}_j \notin \boldsymbol{P}_l} |w_h^{(l,j)}|. \tag{8}$$

The main differences to build ProtoPools (Rymarczyk et al., 2022) from ProtoPNet (C. Chen et al., 2019) are in the following steps:

Dynamical and differentiable assignment of prototypes The authors suggest using the Gumbel-Softmax estimator (Jang et al., 2017; Maddison et al., 2017) Gumbel-Softmax$(q_s, \tau) = (y_s^1, \ldots, y_s^m)$ where $\tau \in (0, \infty)$ is the temperature parameter,

$$y_s^i = \frac{\exp\big((q_s^i + \eta_i)/\tau\big)}{\sum_{j=1}^m \exp\big((q_s^j + \eta_j)/\tau\big)}, \tag{9}$$

and η_j are samples from the standard Gumbel distribution. This estimator generates distributions q_s with exactly one of the probabilities close to 1, which results in a differentiable assignment of the prototypes to classes.

Orthogonal loss The authors add an additional element to the loss function to ensure that the same prototype is not assigned to more than one slot per class,

$$\text{Orth}_p = \sum_{i<j}^{S} \frac{\langle q_i, q_j \rangle}{\|q_i\|_2 \cdot \|q_j\|_2}, \tag{10}$$

which is calculated for each of the different classes.

Projection of prototypes For the prototype projection, the idea is the same as in Equation 7, with the difference that in this occasion

$$\mathcal{Z}_j = \{\tilde{\boldsymbol{z}} : \tilde{\boldsymbol{z}} \in \text{patches}(f(\boldsymbol{x}_i)), \forall i : y_i \in L_j\}, \tag{11}$$

where L_j is the set of classes such that one of its slots corresponds to prototype $\boldsymbol{p}_j$ and the assignment is made by means of the Gumbel-Softmax estimator.

Convex optimization of the last layer In the adaptation to ProtoPools, the authors propose to initialize the weights between the different slots and their assigned class logit to 1, and the rest to 0.

B Experimental setup

Concept-Guided Conditional Diffusion For the Concept-Guided Conditional Diffusion model, we performed experiments for the three types of embedding, sampling images with positive and negative concept combinations. For single-concept guided generations with the CUB dataset, we chose `has_wing_color::black` and `has_wing_color::yellow`, further motivated to explore the effect of the concept imbalance in the training images (41.4% and 6.4%, respectively). Generations with two concepts were made conditioning on `has_wing_color::black` and `has_belly_color::white`, and finally for three concepts we added `has_breast_color::yellow`. For the AwA2 dataset, selected concepts were `black` and `fish` (present in 60.9% and 28.7%), both for generations with one concept and with two concepts. Models were trained on the training split of the CUB dataset (4796 images) and 40% of the images of the AwA2 dataset ($\sim$14292 images). All runs were trained for 1500 epochs and $T = 2000$ steps of diffusion. For the noise scheduler (β_t) we set a linear scheduler with $\beta_1 = 0.0001$ and $\beta_T = 0.2$. The learning rate was set at 0.0003. Models were trained on an NVIDIA GeForce RTX 2080 Ti GPU to generate images of size 64×64.

Concept-Guided ProtoPNet In Concept-Guided Prototype Networks, we measure the concept performance with the test set accuracy across varying architecture configurations, particularly DenseNets, ResNets, and VGGs, and will use the performance of a black-box pre-trained ResNet18 backbone used in the original CBM work (Koh et al., 2020) as an oracle for concept prediction. We generated ten prototypes per each of the $2 \times K$ concept classes (positive and negative). For the prototype datasets, we obtained the 50 closest patches to each prototype, 500 prototype images per concept class. Models were trained for 50 epochs each, with 5 epochs of warm-up and pushing and convex optimization of the last layer every 10 epochs. A summary of the different parameter combinations for the base encoder architecture, the prototype depth and the different coefficients of the loss are available in Table 1.

Table 1: Hyperparameter combinations for Concept-Guided Prototype Network experiments.

Parameter	Values
Base architecture (prototype depth)	VGG16 (128), VGG19 (128), DenseNet121 (128), DenseNet161 (128), ResNet34 (256), ResNet152 (512)
(λ_1, λ_2)	$(0.6, -0.06)$, $(0.8, -0.08)$, $(1, -0.1)$
λ_{last}	10^{-3}, 10^{-4}, 10^{-5}

Concept-Guided ProtoPool Models were trained for 100 epochs with 10 epochs of warm-up. Prototype depth was set as 256 and $\lambda_3 = \lambda_4 = 1$ for both orthogonal losses. The Gumbel-Softmax distribution was initialized with $\tau = 1$ and decreased for 30 epochs. We generated $m = 1000$ prototypes, assigning $S = 10$ slots per concept class. For the concept prototype dataset, we generated 50 samples per prototype, resulting in a maximum of 500 prototypical images (if a prototype is assigned more than once to the same concept class, the resulting number of prototypical images is reduced). The different hyperparameter combinations for base architecture and the rest of the loss coefficients are available in Table 2.

Table 2: Hyperparameter combinations for Concept-Guided ProtoPool experiments.

Parameter	**Values**
Base architecture	DenseNet121, DenseNet161, ResNet34, ResNet50
(λ_1, λ_2)	$(0.6, -0.06)$, $(0.8, -0.08)$, $(1, -0.1)$
λ_{last}	10^{-4}, 10^{-5}

C Results for Concept-Guided Conditional Diffusion

C.1 Generated images for the CUB dataset

Figure 4: Generations for concept `has_wing_color::black` positive.

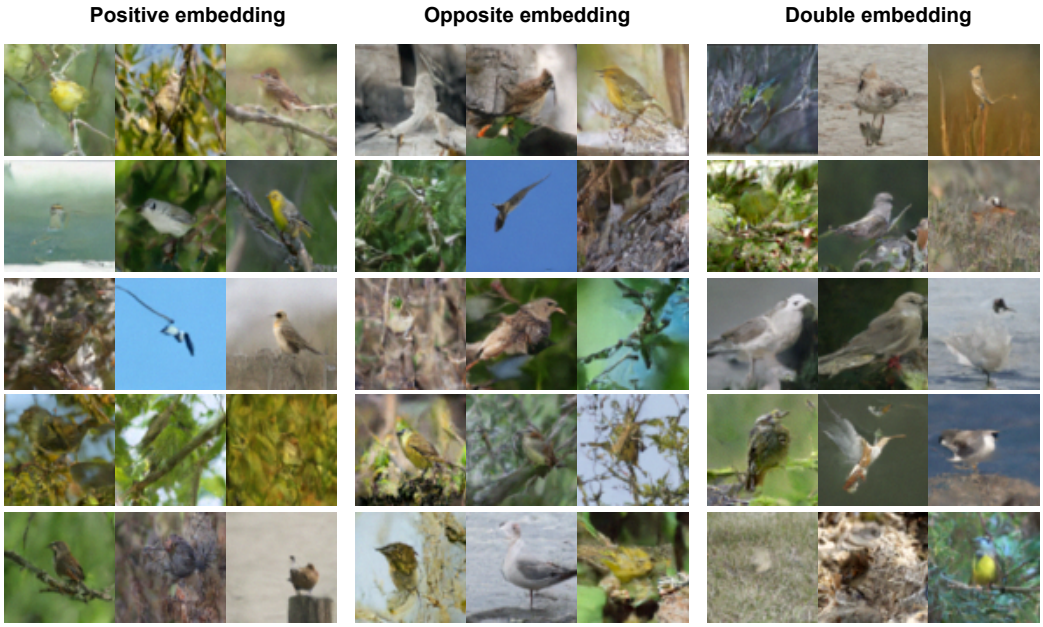

Figure 5: Generations for concept `has_wing_color::black` negative.

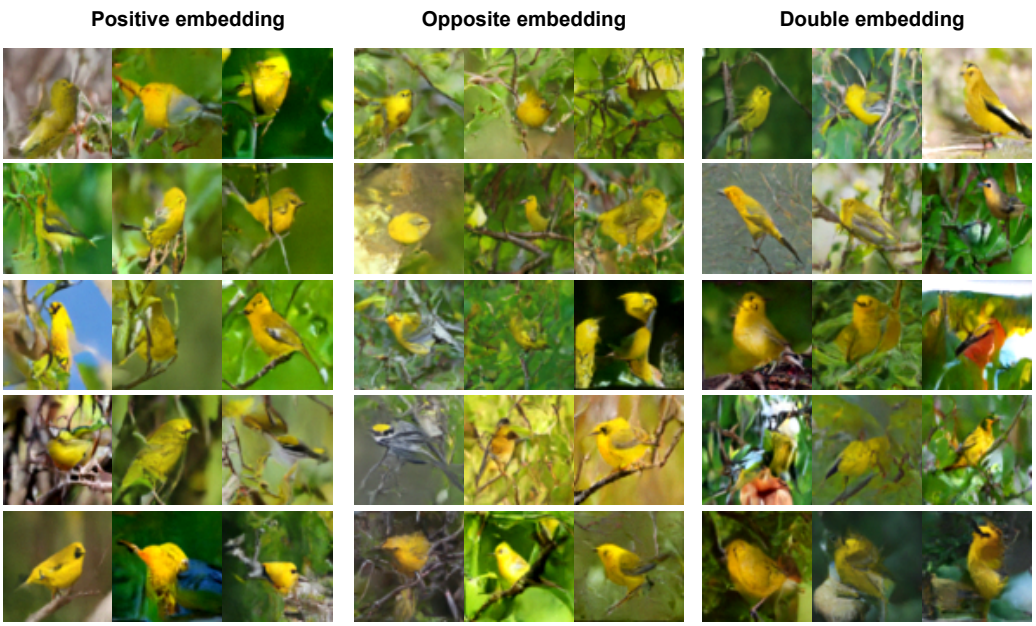

Figure 6: Generations for concept `has_wing_color::yellow` positive.

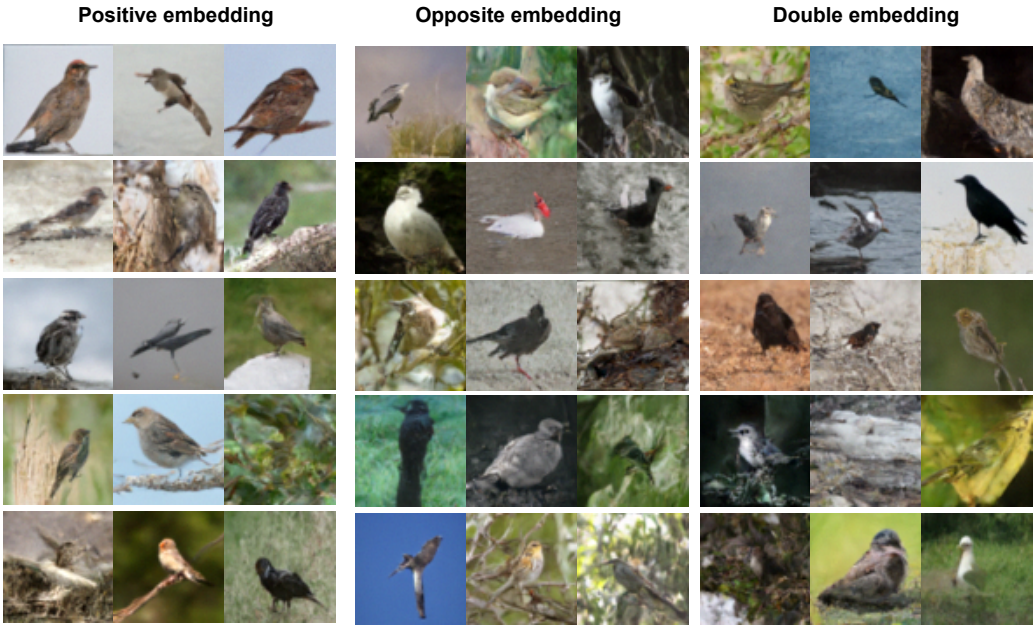

Figure 7: Generations for concept `has_wing_color::yellow` negative.

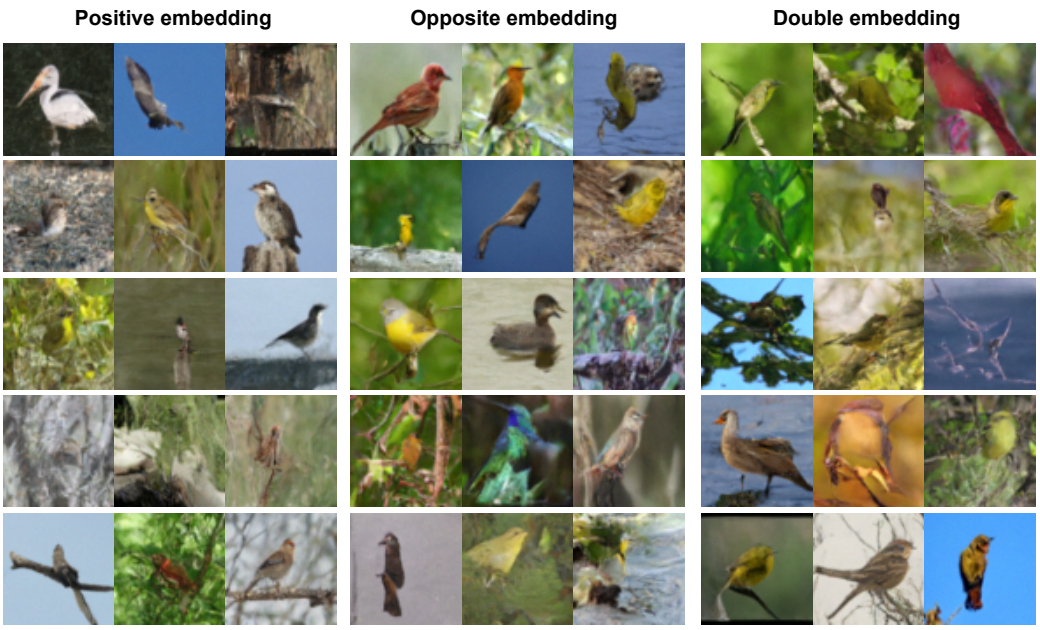

Figure 8: Generations for `has_wing_color::black` and `has_belly_color::white` negative.

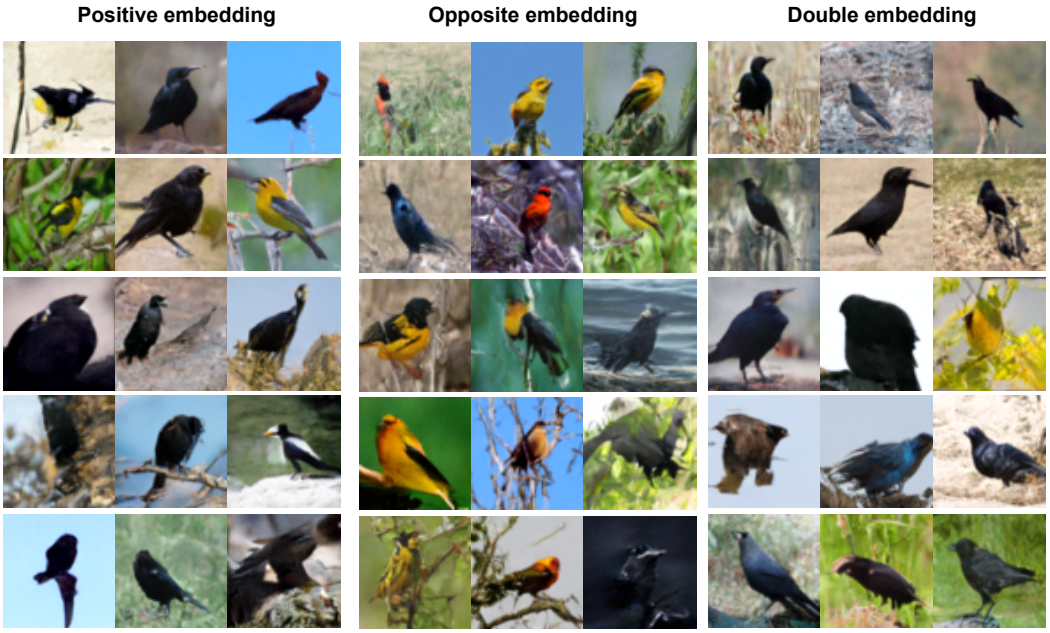

Figure 9: Generations for concept `has_wing_color::black` positive and `has_belly_color::white` negative.

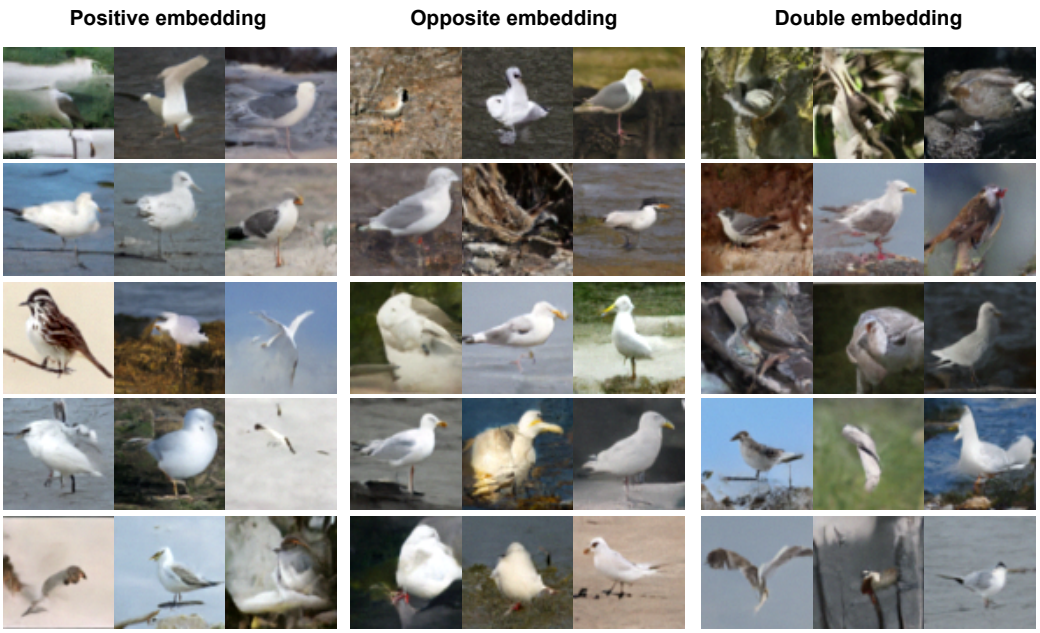

Figure 10: Generations for concept `has_belly_color::white` positive and `has_wing_color::black` negative.

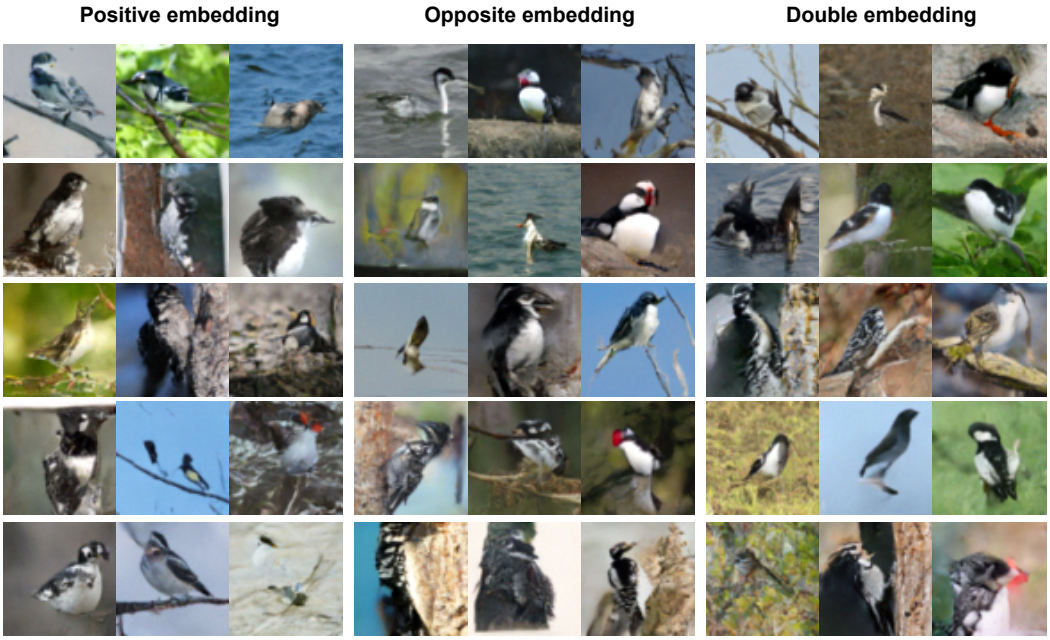

Figure 11: Generations for `has_wing_color::black` and `has_belly_color::white` positive.

Figure 12: Generations for concepts `has_wing_color::black`, `has_belly_color::white` and `has_breast_color::yellow`.

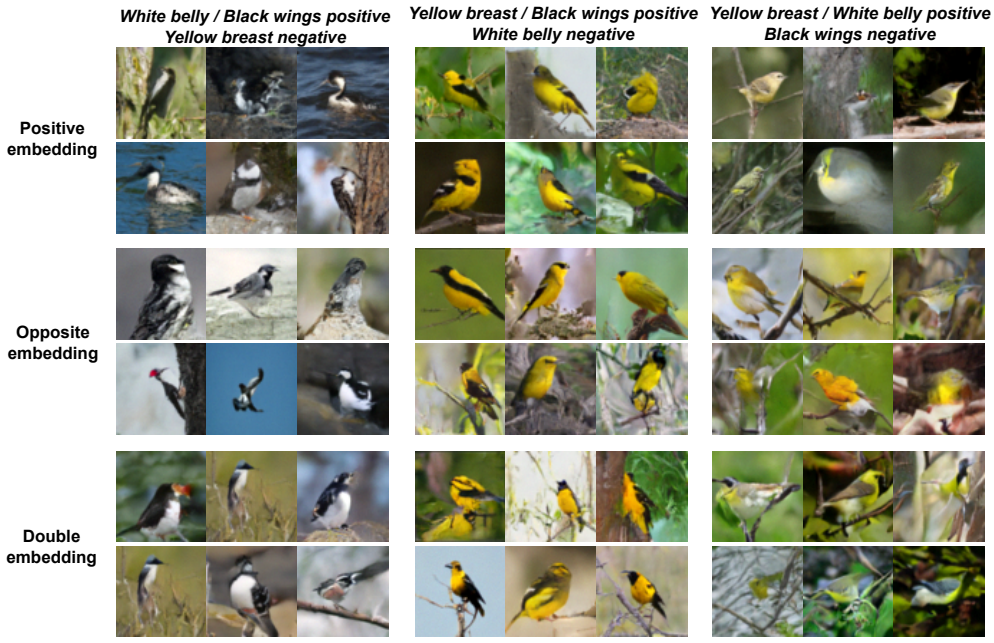

Figure 13: Generations for different combinations of two concepts positive and one negative.

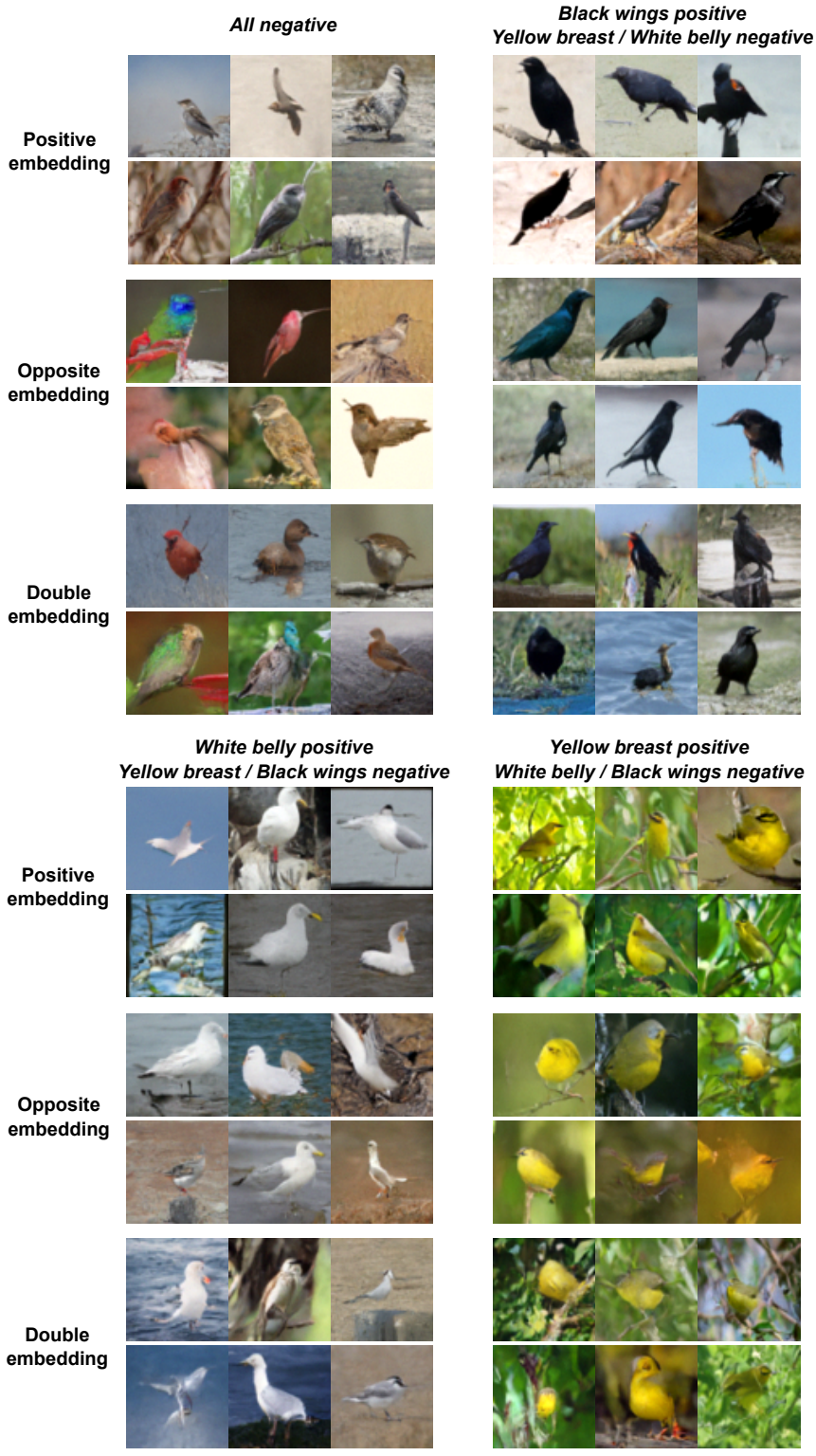

Figure 14: Generations for none of the concepts or just one concept positive.

C.2 Generated images for the AwA2 dataset

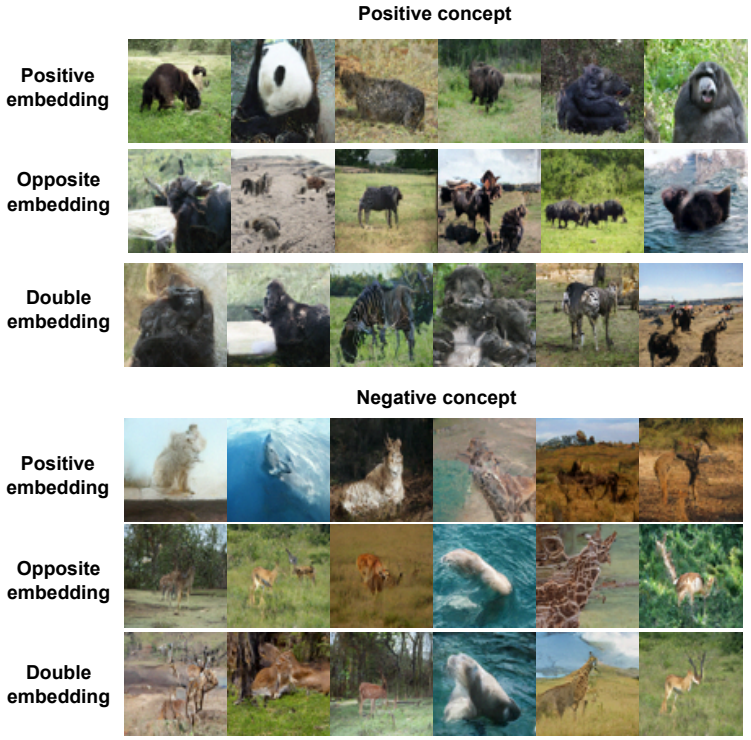

Figure 15: Generations for concept `black` positive and negative.

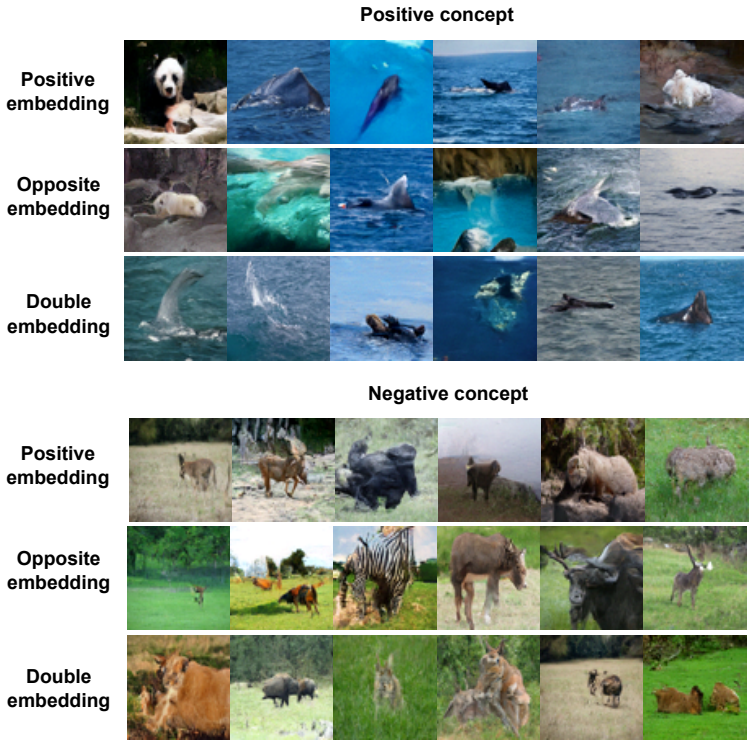

Figure 16: Generations for concept `fish` positive and negative.

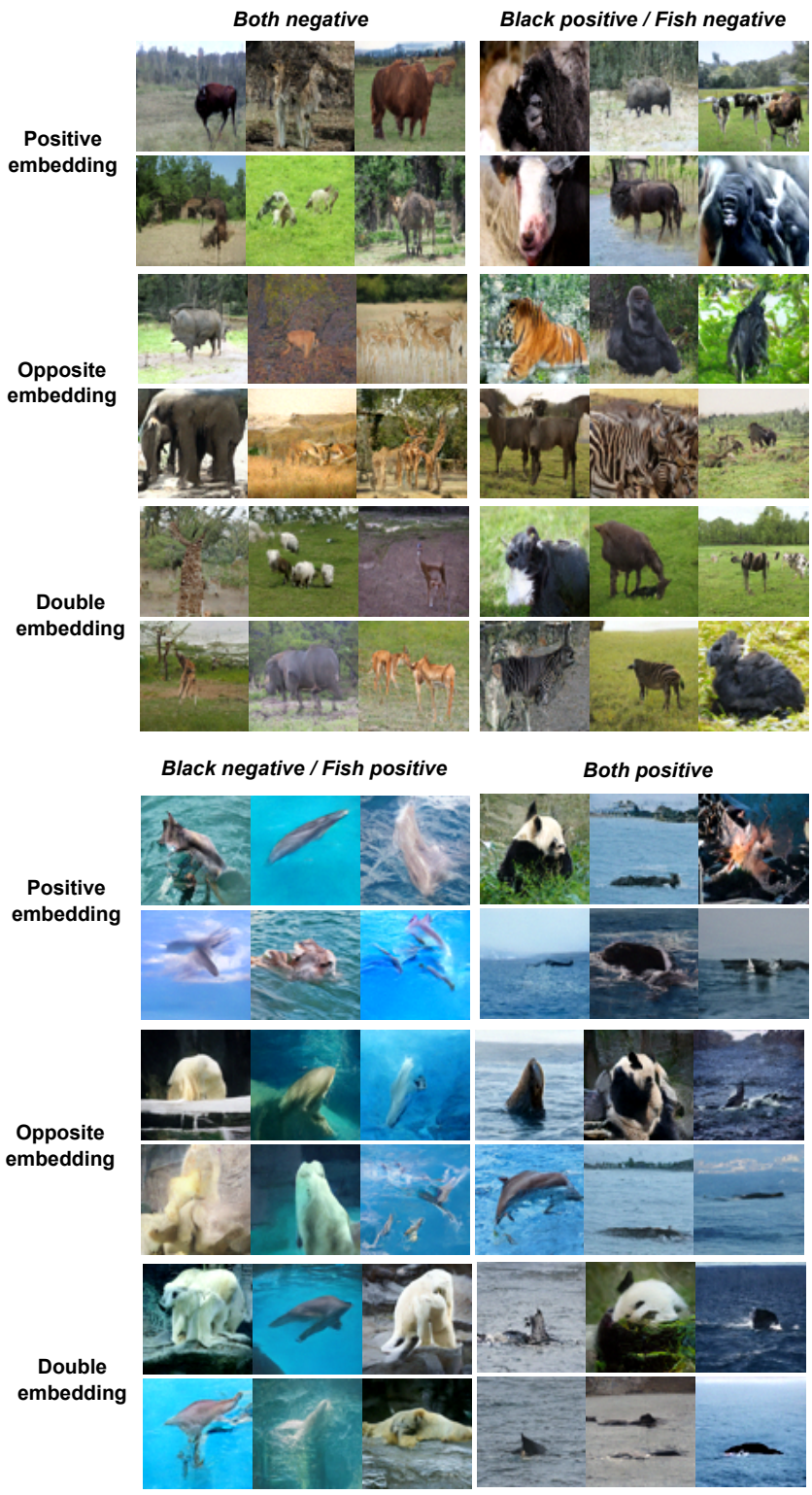

Figure 17: Generations for different combinations of concepts `black` and `fish`.

D Results for Concept-Guided Prototype Networks

Table 3: Hyperparameter tuning accuracy in Concept-Guided ProtoPNet.

Base architecture (prototype depth)	λ_{last}	λ_1, λ_2	**Accuracy (CUB)**	**Accuracy (AwA2)**
Oracle			*0.961*	*0.901*
DenseNet121 (128)	10^{-3}	(0.6, -0.06)	0.864	0.834
		(0.8, -0.08)	0.866	0.813
		(1, -0.1)	0.860	0.836
	10^{-4}	(0.6, -0.06)	0.868	0.821
		(0.8, -0.08)	0.866	0.857
		(1, -0.1)	0.865	0.848
	10^{-5}	(0.6, -0.06)	0.874	0.862
		(0.8, -0.08)	0.874	0.866
		(1, -0.1)	0.872	0.852
DenseNet161 (128)	10^{-3}	(0.6, -0.06)	0.859	0.841
		(0.8, -0.08)	0.859	0.839
		(1, -0.1)	0.845	0.826
	10^{-4}	(0.6, -0.06)	0.865	0.853
		(0.8, -0.08)	0.865	0.829
		(1, -0.1)	0.855	0.821
	10^{-5}	(0.6, -0.06)	0.874	0.866
		(0.8, -0.08)	0.873	0.849
		(1, -0.1)	0.867	0.840
ResNet 34 (256)	10^{-3}	(0.6, -0.06)	0.869	0.830
		(0.8, -0.08)	0.864	0.774
		(1, -0.1)	0.854	0.797
	10^{-4}	(0.6, -0.06)	0.875	0.851
		(0.8, -0.08)	0.870	0.844
		(1, -0.1)	0.869	0.835
	10^{-5}	(0.6, -0.06)	0.878	0.846
		(0.8, -0.08)	0.880	0.841
		(1, -0.1)	0.878	0.834
ResNet152 (512)	10^{-3}	(0.6, -0.06)	0.842	0.821
		(0.8, -0.08)	0.845	0.807
		(1, -0.1)	0.834	0.803
	10^{-4}	(0.6, -0.06)	0.857	0.823
		(0.8, -0.08)	0.859	0.806
		(1, -0.1)	0.863	0.819
	10^{-5}	(0.6, -0.06)	0.867	0.831
		(0.8, -0.08)	0.857	0.828
		(1, -0.1)	0.848	0.836
VGG16 (128)	10^{-3}	(0.6, -0.06)	0.857	0.862
		(0.8, -0.08)	0.851	0.852
		(1, -0.1)	0.852	0.818
	10^{-4}	(0.6, -0.06)	0.865	0.880
		(0.8, -0.08)	0.855	0.871
		(1, -0.1)	0.860	0.868
	10^{-5}	(0.6, -0.06)	0.870	0.884
		(0.8, -0.08)	0.870	0.882
		(1, -0.1)	0.869	0.878

VGG19 (128)	10^{-3}	(0.6, -0.06)	0.854	0.851
		(0.8, -0.08)	0.852	0.840
		(1, -0.1)	0.847	0.837
	10^{-4}	(0.6, -0.06)	0.865	0.867
		(0.8, -0.08)	0.860	0.861
		(1, -0.1)	0.852	0.831
	10^{-5}	(0.6, -0.06)	0.871	0.881
		(0.8, -0.08)	0.873	0.865
		(1, -0.1)	0.868	0.864

Table 4: Hyperparameter tuning accuracy in Concept-Guided ProtoPool.

Base architecture (prototype depth)	λ_{last}	λ_1, λ_2	**Accuracy (CUB)**	**Accuracy (AwA2)**
Oracle			*0.961*	*0.901*
DenseNet121	10^{-4}	(0.6, -0.06)	0.817	0.880
		(0.8, -0.08)	0.816	0.877
		(1, -0.1)	0.814	0.877
	10^{-5}	(0.6, -0.06)	0.867	0.873
		(0.8, -0.08)	0.860	0.879
		(1, -0.1)	0.854	0.872
DenseNet161	10^{-4}	(0.6, -0.06)	0.815	0.880
		(0.8, -0.08)	0.816	0.880
		(1, -0.1)	0.815	0.878
	10^{-5}	(0.6, -0.06)	0.870	0.876
		(0.8, -0.08)	0.877	0.878
		(1, -0.1)	0.873	0.878
ResNet34	10^{-4}	(0.6, -0.06)	0.812	0.870
		(0.8, -0.08)	0.813	0.861
		(1, -0.1)	0.814	0.859
	10^{-5}	(0.6, -0.06)	0.878	0.865
		(0.8, -0.08)	0.878	0.865
		(1, -0.1)	0.874	0.860
ResNet50	10^{-4}	(0.6, -0.06)	0.814	0.887
		(0.8, -0.08)	0.814	0.888
		(1, -0.1)	0.814	0.888
	10^{-5}	(0.6, -0.06)	0.860	0.883
		(0.8, -0.08)	0.860	0.890
		(1, -0.1)	0.853	0.889

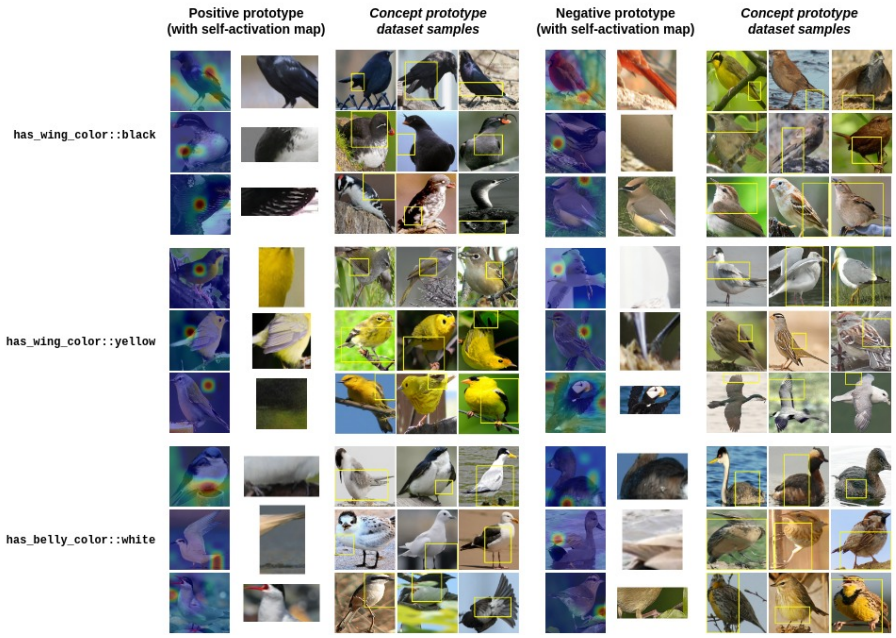

Figure 18: Positive and negative prototypes with self-activation map and samples from the concept dataset (yellow square over the original image) for Concept-Guided ProtoPNet in the CUB dataset.

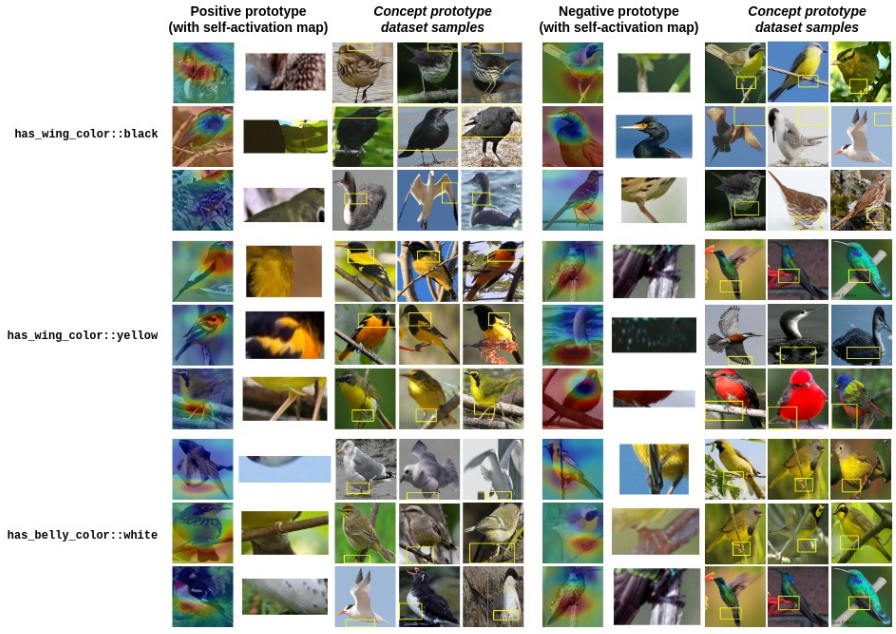

Figure 19: Positive and negative prototypes with self-activation map and samples from the concept dataset (yellow square over the original image) for Concept-Guided ProtoPools in the CUB dataset.

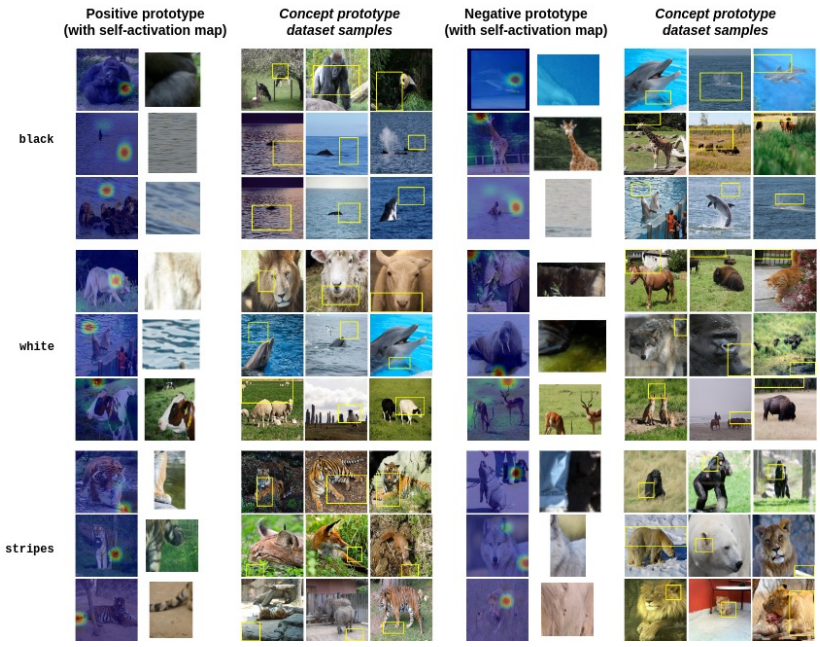

Figure 20: Positive and negative prototypes with self-activation map and samples from the concept dataset (yellow square over the original image) for Concept-Guided ProtoPNet in the AwA2 dataset.

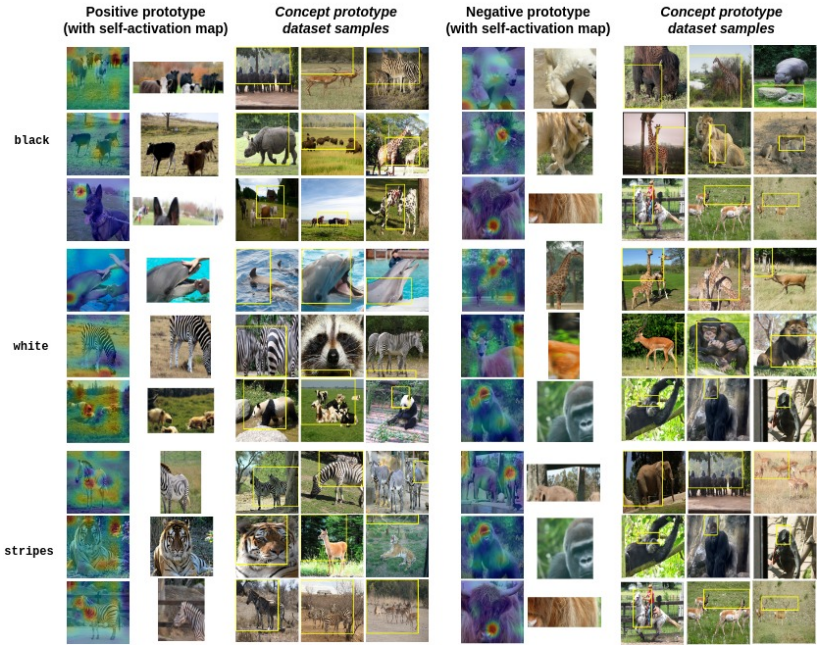

Figure 21: Positive and negative prototypes with self-activation map and samples from the concept dataset (yellow square over the original image) for Concept-Guided ProtoPools in the AwA2 dataset.