DAPPNet: Several Tricks to Augment Deformable ProtoPNet

Zixuan Cao[1,2] and Xiang Ji[1,3]

[1]IIIS, Tsinghua University
[2]2022011336
[3]2022010881
{caozx22, jix22}@mails.tsinghua.edu.cn

Abstract

The deformable prototypical part network (Deformable ProtoPNet, DPPN) is an interpretable image classifier that combines the strengths of deep learning and the interpretability of case-based reasoning, as proposed by Jon Donnelly et al. DPPN uses prototypes consisting of spatially deformable prototypical parts to present a flexible, interpretable and convincing image in training dataset that "looks like" the input image. However, there is room for further enhancement. We identified four primary issues with the initial model and implemented four specific tricks to address these shortcomings. Our ***Deformable Augmented Prototypical Part Network (DAPPNet)*** *not only boosted the model's accuracy but also substantially enriched the interpretability and detailed representation of visual features, enhancing overall model effectiveness.*

1. Introduction

"How would you describe why an image looks like a clay colored sparrow?" This question, posed by Chaofan Chen et al. in [4], highlights a long-standing challenge: neural networks are often seen as black boxes. The non-linearity and multitude of parameters in these models obscure their inner workings, making it difficult for humans to understand why and how images are classified. However, the application of deep learning extends beyond requiring high accuracy; it also demands robust interpretability. In critical fields such as medicine [2, 17], finance [23], and criminal justice [3], stakeholders need clear and substantial reasoning to trust and rely on these systems.

1.1. Interpretable Neural Networks

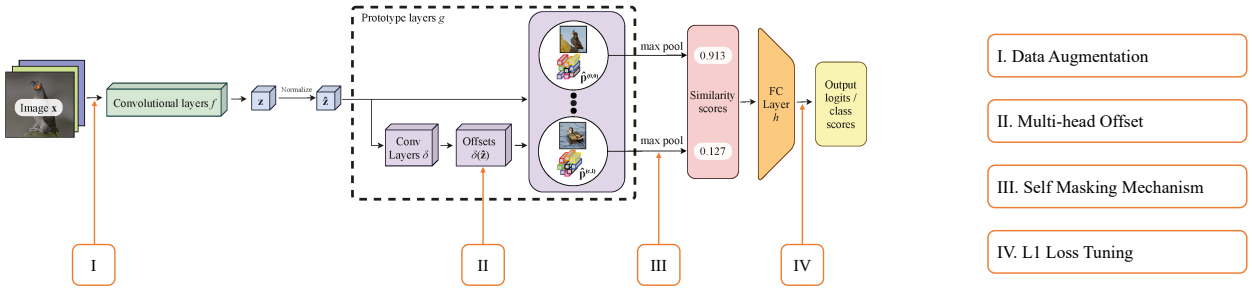
To tackle this issue, two approaches have been proposed: (1) posthoc methods [1, 13, 25] and (2) case-based methods. Posthoc methods derive interpretability from the model's predictions. However, these methods do not truly explain the decision-making process, as they generate explanations only after making inferences. In contrast, case-based methods develop inherently interpretable neural networks that are capable of self-explanation. Unlike approaches that retroactively infer the thought process from the outcomes, case-based models achieve genuine interpretability by using specific cases to illustrate how they classify test images.

Before Chaofan Chen *et al*. [4, 7, 11], there were two main paradigms: *Prototype Trees(ProtoTree)* [10] and the *Bayesian Case Model (BCM)* [16]. ProtoTree merges prototype learning with decision trees, creating a model that is globally interpretable by design. Additionally, ProtoTree can provide a localized explanation for an individual prediction by tracing a decision path through the tree. On the other hand, the BCM approach learns prototypes through joint inference on cluster labels, prototypes, and key features. It also emphasizes sparsity by identifying subspaces which is the sets of features that are crucial for defining the prototypes.

1.2. The Trilogy of Prototype Part Network

From 2017 to 2024, Oscar Li, Chaofan Chen, Jon Donnelly *et al*. introduced a progressive series of prototype-based deep learning methods. These include: (1) an Autoencoder and prototype-based network [11], (2) the Prototype Part Network (ProtoPNet) [4], and (3) the Deformable ProtoPNet (DPPN) [7]. Across this trilogy of developments, the authors systematically enhanced both the interpretability and the accuracy of the networks, demonstrating an evolutionary approach and illuminating the development direction of prototype-based learning.

The Autoencoder and Prototype-based Network [11] merges an autoencoder with a prototype layer, where each unit stores a weight vector resembling an encoded training input. The encoder extracts features from test images and



(a) Vanilla architecture of DPPN.

(b) Our improving tricks.

Figure 1. Illustration of DPPN. (a) the illustration of original DPPN pipeline, coming from the original paper[7] (b) the improving tricks proposed by our work.

the decoder visualizes the learned prototypes. Similarity between a test image and the prototypes is computed as follows:

$$p(\mathbf{z}) = \left[\|\mathbf{z}-\mathbf{p}_1\|_2^2, \|\mathbf{z}-\mathbf{p}_2\|_2^2, \ldots, \|\mathbf{z}-\mathbf{p}_m\|_2^2\right]^T$$

$$s(\mathbf{v})_k = \frac{\exp(v_k)}{\sum_{k'=1}^{K}\exp(v_{k'})}$$

where v_k is the k-th component of the vector $\mathbf{v} = \mathbf{W}p(\mathbf{z}) \in \mathbb{R}^K$.

However, the network is currently implemented on the Fashion MNIST dataset [24], and the simple autoencoder architecture may be inadequate for more complex data in terms of feature extraction and visualization of the thought process. Therefore, a ProtoPNet utilizing existing CNN backbones is proposed [4]. The features extracted by the CNN are compared to the features of prototypes, and visualization is achieved by upsampling to identify the most activated part of the test image. The similarity is calculated using the L_2 -norm, as outlined in the main theorem of [4]. Furthermore, the paper introduces an innovative training procedure comprising three stages.

ProtoPNet has sparked debate regarding its effectiveness. Critics contend that images perceived as similar by humans may not share analogous features in latent space, a vulnerability to attacks like PGD [8]. This issue is further explored in [9], which introduces two evaluation metrics, "consistency score" and "stability score," to bridge the semantic gap between feature space and input space. Additionally, some variants of ProtoPNet enhance prototype-based image classification by incorporating multi-patch visualizations for improved conceptual clarity and quantifying visual characteristics to bolster interpretability and accuracy [14, 15].

Motivated by the Deformable Convolutional Networks introduced by Jifeng Dai *et al*. [5], developments in deep metric learning using cosine or angular margins [6, 21], and the emerging trend of contrastive learning, Jon Donnelly *et al*. proposed Deformable ProtoPNet (DPPN) [7]. DPPN integrates methodologies from [22], [5], and [4]. While prior methods employed spatially rigid prototypes, DPPN overcomes this limitation by introducing spatially flexible prototypes. Moreover, DPPN has achieved state-of-the-art accuracy on the CUB-200-2011 bird recognition dataset [20].

1.3. Architecture and Main Concepts in DPPN

The main architecture of the vanilla DPPN can be found in Figure1a. As we introduced before, DPPN uses ResNet, VGGNet or DenseNet as CNN backbone f. The extracted feature $\mathbf{z} = f(\mathbf{x})$ is a tensor with shape $\eta_1 \times \eta_2 \times d$. Then the feature will be compared with $c \times l$ prototypes in the prototype layer g, where $c = 200$ is the number of classes and $l = 6$ is the number of prototypes for each class saved in the prototype layer. Each prototype $\mathbf{p}^{(c,l)}$ is of shape $\rho_1 \times \rho_2 \times d$. The similarity of the input feature $\mathbf{z}$ and a prototype $\mathbf{p}^{(c,l)}$ is defined as

$$g(\mathbf{z})_{a,b}^{(c,l)} = \text{sim}\left(\sum_m \sum_n \|\mathbf{p}_{m,n}^{(c,l)} - \mathbf{z}_{a+m,b+n}\|^2\right).$$

Note that if we apply normalization in the feature space and set a particular similarity function, the similarity score can be

$$g(\hat{\mathbf{z}})_{a,b}^{(c,l)} = \sum_m \sum_n \hat{\mathbf{p}}_{m,n}^{(c,l)} \cdot \hat{\mathbf{z}}_{a+m,b+n},$$

which is simply the cosine angle of the 2 tensors.

However, this cosine angle is rigid as in ProtoPNet. To apply deformable prototypes, we use deformable convolution in [5]. The offset of a position (a, b) in $\mathbf{z}$ is computed by a network $\delta(a, b, \mathbf{z}) = (\Delta_1, \Delta_2)$. Therefore the complete similarity score in DPPN is

$$g(\hat{\mathbf{z}})^{(c,l)}_{a,b,m,n} = \hat{\mathbf{p}}^{(c,l)}_{m,n} \cdot \hat{\mathbf{z}}_{a+m+\Delta_1, b+n+\Delta_2}.$$

And sim-score at each position (a, b) will be taken the maximum, to get the score of how confident the model think this prototype is in this image:

$$g(\hat{\mathbf{z}})^{(c,l)} = \max_{a,b} g(\hat{\mathbf{z}})^{(c,l)}_{a,b},$$

where $g(\hat{\mathbf{z}})^{(c,l)}$ is the output on each proto-neuron of prototype layer.

Finally, the output of prototype layer will be used to compute confidence for each class using a fully connected layer h. We then take soft-max for $h \circ g \circ f(\mathbf{x})$.

1.4. Training Algorithm and Learning Object for DPPN

The training procedure comprises 3 stages, the same as in [4]:

(1) **SGD Process:** Apply Stochastic Gradient Descent (SGD) on f, g, and δ. To initialize, randomly set $\mathbf{p}$ in $(0, 1)$, and set $h^{((c,l),c)} = 1, h^{((c,l),c')} = -0.5$. At this step, learning object consists of several parts:

- Cluster loss encourages proto-neurons to learn a better feature for the same class.

$$\ell_{\text{cst}} = -\frac{1}{N} \sum_{i=1}^{N} \max_{\hat{p}_{c,l}: c=y(i)} g(\hat{\mathbf{z}}^{(i)}_{c,l})$$

- Separation loss punishes the model for a high score from other classes.

$$\ell_{\text{sep}} = \frac{1}{N} \sum_{i=1}^{N} \max_{\hat{p}_{c,l}: c\neq y(i)} g(\hat{Z}^{(i)}_{c,l})$$

From a personal perspective, the two losses mentioned above incorporate elements of contrastive learning.

- Cross-entropy loss assesses the model for overall prediction.

$$CE^{-} = \sum_{i=1}^{N} -\log \frac{\exp\left(\sum_{c,l} w_h^{((c,l),y(i))} g^{(-)}(i)^{c,l}\right)}{\sum_{c'} \exp\left(\sum_{c,l} w_h^{((c,l),c')} g^{(-)}(i)^{(c,l)}\right)},$$

where $g^{(-)}$ is particularly designed as

$$g^{-}(i)^{(c,l)} = \begin{cases} g(\hat{\mathbf{z}}^{(i)})^{(c,l)} & \text{if } c = y^{(i)}, \\ \max_{a,b} \cos\left(\lfloor \theta(\hat{\mathbf{p}}^{(c,l)}, \hat{\mathbf{z}}^{\Delta,(i)}_{a,b}) - \phi \rfloor_+\right) & \text{else.} \end{cases}$$

This idea of margin comes from [6, 12, 21].

- Orthogonality loss encourages prototypical parts within the same class and among the same prototype to be orthogonal to other.

$$\ell_{\text{ortho}} = \sum_{c} \left\| \mathbf{P}^{(c)} \mathbf{P}^{(c)\top} - r^2 \mathbf{I}^{(\rho_L)} \right\|_F^2, \quad (1)$$

where $\mathbf{P}$ is the $\rho L \times d$ matrix consists of prototypical parts.

So the complete learning object is

$$\ell = \text{CE}^{(-)} + \lambda_1 \ell_{\text{sep}} + \lambda_2 \ell_{\text{cst}} + \lambda_3 \ell_{\text{ortho}}$$

(2) **Push Process:** Push prototype vectors to a real prototype in proto-neurons g.

$$\mathbf{p}^{(c,l)} \leftarrow \arg\max_{\mathbf{z}^{(\Delta)}_{a,b}} \cos\left(\theta(\hat{\mathbf{p}}^{(c,l)}, \hat{\mathbf{z}}^{(\Delta)}_{a,b})\right)$$

(3) **Convex Optimization of h:** Optimize h. This is a convex optimization.

$$\ell_{\text{last}} = \text{CE} + \lambda_1 \ell_l,$$

where ℓ_1 is specified in the code as

$$\ell_1 = \sum_{c,l} \sum_{c'\neq c} \left| w_h^{((c,l),c')} \right|. \quad (2)$$

The ℓ_1 loss comes from the main theorem in ProtoPNet [4], which requires $w_h^{(k,j)} = 1$ for j with $\mathbf{p}_j \in \mathbf{P}_k$ and $w_h^{(k,j)} = 0$ for j with $\mathbf{p}_j \notin \mathbf{P}_k$.

2. Issue Proposal

While the series of ProtoPNet introduced by previous researchers represents a significant and innovative contribution to the field, there remains room for refinement and enhancement. In the following section, we will explore some of the potential limitations identified in their approach.

Issue #1. The gap. We identify a huge gap between train and test accuracy when the training process converges, up to 14% (refer to Figure 2). This may be led by overfitting on the training set, since we use a powerful model ResNet50, on a relatively small dataset with only 5000+ training image.

Issue #2. I can see myself! Another factor contributing to the substantial gap and difficulty in improving test accuracy may relate to self-reference during training. Specifically, for a prototype neuron that stores a prototype vector derived from image i, during subsequent epochs post-pushing, when the model classifies training image i, it recognizes its own prototype in the latent space. This self-recognition results in artificially high confidence for that class. Consequently, this diminishes the Backpropagation

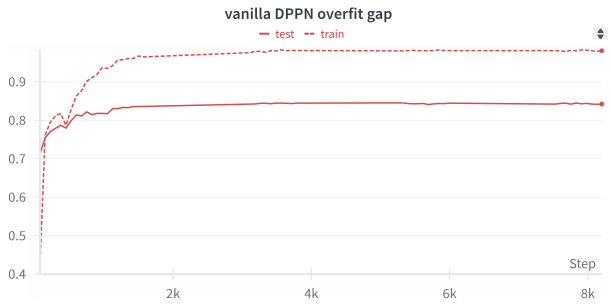


Figure 2. Overfit Gap of Vanilla DPPN.

algorithm's ability to learn effectively from other prototype neurons. Moreover, once the model develops high confidence in certain prototype neurons, it becomes less likely to update these neurons with new prototypes, stalling further learning.

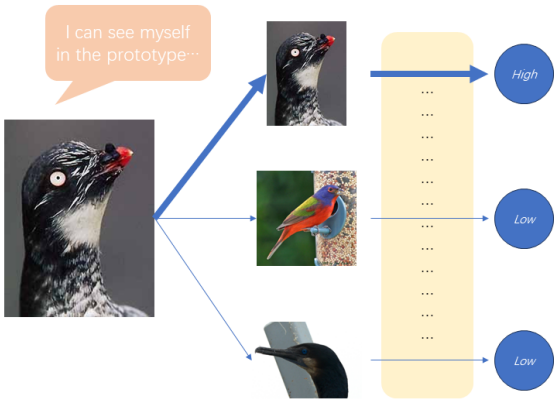


Figure 3. I can see myself!

Issue #3. Oversimplified Offset Prediction. In the vanilla implementation of DPPN, the offset-predicting network δ comprises just two convolutional layers. This offset function is solely determined by the dimensions of the feature space $\rho_1 \times \rho_2$. Consequently, regardless of the specific prototype or its position (a, b) within the feature space, the offset Δ for a given position remains constant (refer to Figure 4). As a result, despite the intended "deformability" of the prototypes in DPPN, they exhibit a level of "rigidity" that limits the model's adaptability and effectiveness.

Issue #4. An inconsistency between theory and practice. We observe a notable discrepancy between the theoretical expectations and practical implementation within the ProtoPNet and Deformable ProtoPNet frameworks. The main theorem in [4] establishes that to maintain minimal difference of logit, four specific assumptions or requirements must be met, where the fourth requirement states:

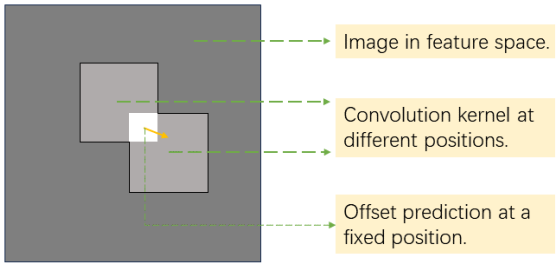


Figure 4. The patch in different convolution kernels corresponds to a same offset.

> (A4) For each class k, the weight connection in the fully connected last layer h between a class k prototype and the class k logit is 1, and that between a non-class k prototype and the class k logit is 0 (i.e., $w_h^{(k,j)} = 1$ for all j with $\mathbf{p}_j \in \mathbf{P}_k$ and $w_h^{(k,j)} = 0$ for all j with $\mathbf{p}_j \notin \mathbf{P}_k$).

However, in practice, the implementation of this requirement involves applying an ℓ_1 loss to optimize h, as indicated in Equation 2. This implementation not only penalizes $w_h < 0$ but also $w_h > 1$, potentially leading to excessive restriction, which could diverge from the intended theoretical framework.

3. Main Improvements and Experiments

Following our identification of issues within Deformable ProtoPNet 2, we have implemented several strategic improvements to address accuracy, interpretability, richness, and completeness. This led to the development of **Deformable Augmented Prototypical Part Network (DAPPNet)**, which realizes these enhancements to better align theoretical concepts with practical applications. All these tricks are illustrated in Figure 1b, and the effect of our improvements is visualized in Appendix A.

Trick I. Data Augmentation. Addressing *Issue #1* previously identified, we aim to narrow the accuracy gap between training and testing phases. Given the complex training algorithm and multiple hyperparameters of DPPN, implementing data augmentation requires dalicate consideration. PPNet [4] employs offline data augmentation. We explored two types of online data augmentation in our experiments. The first approach utilized severe augmentation techniques, while the second adopted milder methods, omitting transformations such as color and brightness changes.

Trick II. Multi-headed Offset. As illustrated in Figure 4, the standard offset predicting network δ in the vanilla implementation only assigns a single offset to each patch in the

feature space. The fundamental concept behind deformable convolution is to allow patches to shift to non-integer positions, thereby enhancing representational power and interpretability. However, this implementation remains somewhat "rigid". For example in Figure 5, consider a patch initially located at the neck of a bird; if one prototype necessitates a shift to the head and another to the wings, the single-headed offset approach (i.e., vanilla) cannot accommodate such diverse requirements. This limitation underscores the need for a more flexible offset mechanism to learn more features.

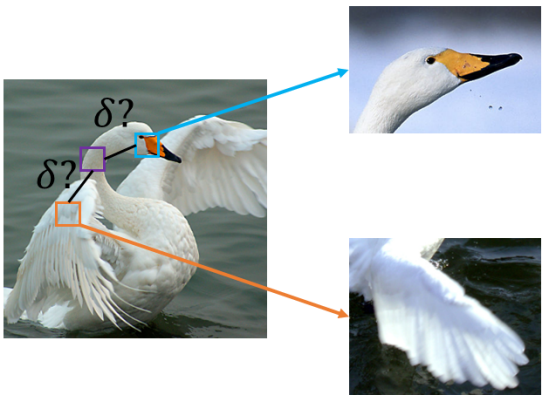


Figure 5. Which offset to learn?

In our experiment, we implemented a multi-headws offset, to encourage more offset-shifting possibility, rather than a single direction. This work is motivated by multi-head attention applied in natural language processing [19]. As shown in Figure 6, the offset-predicting network δ provide more offsets, each of which learns one suit of offsets. When a prototype neuron is applied to the image, all the offsets will be tired to compute similarity score for the prototype. Then we take maximum the scores, as the input the FCN h.

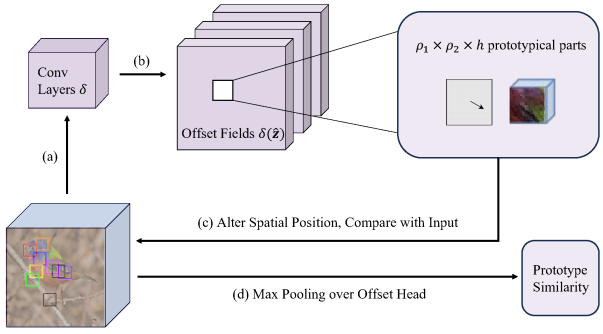


Figure 6. Multi-headws offset.

Trick III. Self-Masking Mechanism. According to *Issue #3*, detailed in Figure 3 and discussed in Section 2, if a training image recognizes itself in the prototype neurons, the model develops high confidence in its prediction. However, this scenario does not occur during the testing process. To address this discrepancy, we propose a self-masking mechanism that applies a mask to the prototypes when a training image "see itself". This approach encourages the prediction process to rely on a broader array of prototypes, rather than depending solely on one.

Note that when a prototype is masked, predictions will depend on only the remaining five or fewer prototypes in the class. This creates an imbalance, as unmasked classes benefit from all six prototypes. This issue is particularly critical in our implementation, where all repeated images in a batch are masked. Therefore, the final score of a class should be adjusted to reflect the reduced number of contributing prototypes:

$$final\ score^{(c)} = \frac{\#\ total\ prototypes}{\#\ left\ prototypes^{(c)} + \epsilon} \times original\ score^{(c)}.$$

In this way, we ensure fairness and consistency across classifications.

Trick IV. L1 Loss Tuning. As highlighted in *Issue #4*, there is a discrepancy between the primary theoretical framework of [4] and its practical implementation. Specifically, the use of ℓ_1 loss leads to the over-penalization of weights in the last layer. This method unnecessarily restricts weights to non-positive values according to the theorem, prompting us to propose a modified loss function as a remedy. This new function contrasts with the original version as shown in Equation (2):

$$\ell_1' = \sum_{c,l} \sum_{c' \neq c} \mathrm{ReLU}\left(-w_h^{((c,l),c')}\right). \tag{3}$$

4. Results

4.1. Training Results

We applied our techniques to the Deformable ProtoPNet framework, and it appears that our methods enhance the performance of the original model in several ways. Figure 7 presents the overall training results on the CUB-200-2011 dataset [20], which is the benchmark used by the original authors of DPPN. In this figure, each modification corresponds to a previously proposed technique, with all modifications implementing mild data augmentation except for the "Severe Data Augmentation" setting.

4.2. Explanation

From the training curve we can discuss with the effect of our tricks. Honestly, they don't work outstandingly well, but there is actually a lot of interesting findings.

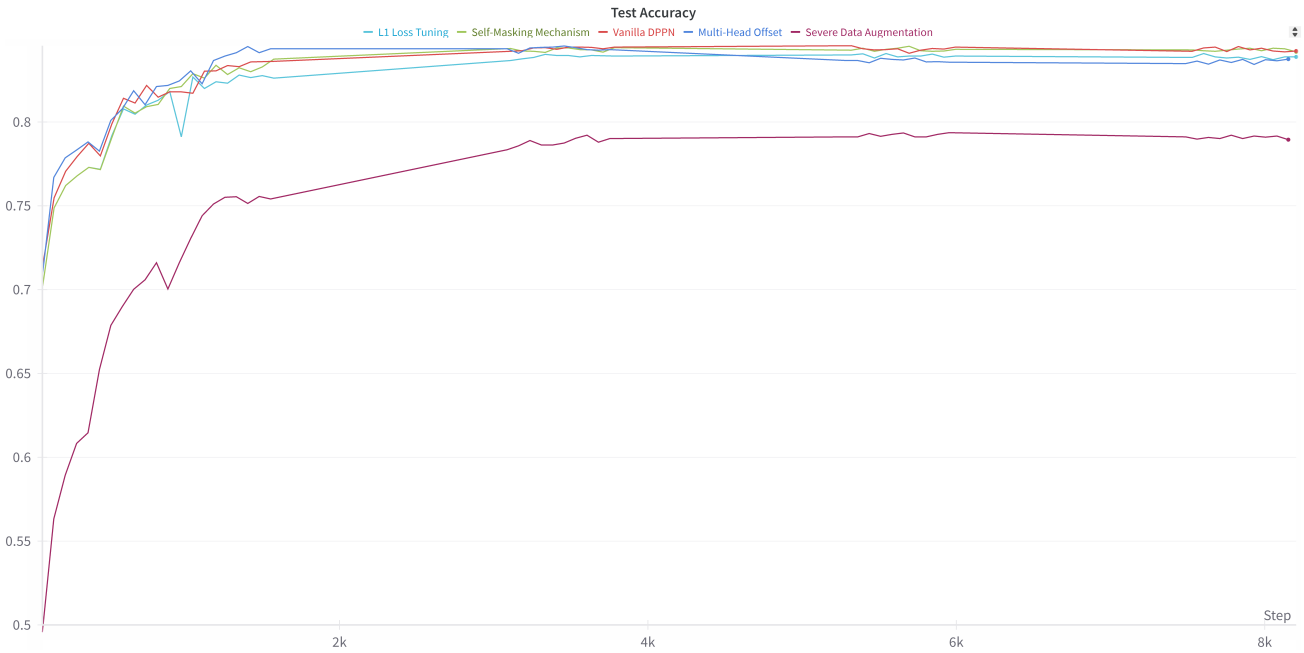


Figure 7. Test Accuracy of Each Implementation.

Method	10 epochs	20 epochs	30 epochs	40 epochs	50 epochs
Vanilla DPPN	81.48%	83.98%	**84.33%**	84.31%	84.25%
Severe Data Aug.	71.60%	78.52%	79.14%	79.17%	78.53%
Multi-headed Offset	**82.12%**	84.07%	80.55%	83.28%	82.53%
Self-Masking Mechanism	81.05%	**84.27%**	84.30%	**84.40%**	**84.41%**
L1 Loss Tuning	81.30%	83.58%	83.95%	83.97%	83.87%

Table 1. Comparison of Test Accuracy (after * epochs and after pushing).

Trick I. Data Augmentation. Compared with both the original implementation and overly aggressive data augmentation strategies, our experimental results suggest that gentle data augmentation helps the model balance efficiency and generalizability more effectively. The model subjected to severe data augmentation converges with an accuracy approximately 5 percentage points lower than that achieved by the milder approach.

Our analysis indicates that excessively severe data augmentation does not enhance test accuracy. Given that DPPN uses training images as prototypes, highly altered prototypes may fail to capture essential features accurately. Moreover, in fields such as ornithological systematics, the accurate representation of a bird's color is crucial; thus, modifying color and brightness excessively during training proves counterproductive, as evidenced by the superior performance of milder data augmentation.

Trick II. Multi-headed Offset. We implemented a 4-head variant in our experiments. Initially, our multi-headed version outperformed the standard model by approximately one percent. However, as training progressed, the performance advantage diminished.

This convergence in performance may be attributed to the diminishing returns of additional model complexity. Initially, the multiple heads in our model can learn diverse and complementary features, providing a rapid improvement. Over time, however, as the dataset's core features become thoroughly learned, the additional heads offer less incremental benefit, and the complexity of coordinating these heads may lead to inefficiencies that allow the simpler, vanilla model to catch up.

Trick III. Self-Masking Mechanism. We incorporated a self-masking mechanism into our model to discourage overfitting and encourage robustness. Initially, our self-masking variant lagged behind the vanilla model. Yet, as training continued, it caught up with the standard model, demonstrating a behavior opposite that of the multi-headed offset.

This phenomenon likely results from the characteristics of the masking operation. Since it is only applied during the pushing phase, it has little impact when no pushing is done, or when the "I can see myself" paradox rarely occurs early in training when the model is still randomly initialized. Therefore, the self-masking mechanism acts as a long-term tuning tool, particularly useful for addressing mode collapse.

Trick IV. L1 Loss Tuning. We substituted the standard ℓ_1 loss with a modified objective in our experiments. As expected, this alteration did not lead to substantial improvements. From an optimization perspective, the original method functions effectively as a special case of lasso regression [18], applying regularization to the weights of the final layer.

5. Conclusion and Discussion

In this project, we developed DAPPNet, which enhances DPPN across several dimensions: accuracy, interpretability, richness, and completeness. We identified four key issues and introduced strategic solutions to address them. While $Trick1$ data augmentation and $Trick4$ ℓ_1 loss tuning did not yield substantial results, the underlying reasons are discussed in detail in Section 3. However, the outcomes from $Trick2$ multi-headed offset and $Trick3$ self-masking mechanism were notably positive, demonstrating the effectiveness of our approaches.

However, several intriguing issues and observations still remain, awaiting further analysis and implementation. We will explore these in the following discussion.

5.1. The Orthogonal Loss

In DPPN [7], the author introduced an innovative learning object in the first stage of training procedure 1,

$$\ell_{\text{ortho}} = \sum_{c} \left\| \mathbf{P}^{(c)} \mathbf{P}^{(c)\top} - r^2 \mathbf{I}^{(\rho_L)} \right\|_F^2 .$$

Ideally, this loss term encourages prototypical parts within a class to be orthogonal to each other, thereby prompting the model to learn distinct features. However, this approach may not always be effective. For instance, if two prototypes within the same class both initially learn features of wings and are then forced to diverge, it could result in neither prototype effectively capturing this crucial feature.

We can address this issue by modifying the loss term to avoid penalizing two prototypical parts if they are sufficiently similar. Otherwise, we continue to encourage separation as before. Intuitively, if two prototypes learn the same feature, this feature is likely crucial for the class. Moreover, redundancy in prototypes does not significantly impact the prediction process, as the last layer can adjust to recognize the same part multiple times.

5.2. Self-Masking Mechanism

In Section 3, we introduced the self-masking mechanism to prevent training images from recognizing themselves in prototype neurons. However, this approach inadvertently lowers the expected score for the masked class. For a test image, the total score comprises contributions from 6 prototypes, but for a masked training image, the score comes from at most 5 prototypes. To correct this skewed expectation, we can adjust the remaining score by multiplying it by the fraction of the class that is masked. The adjusted expected score is calculated as follows:

$$final\ score^{(c)} = \frac{\#\ total\ prototypes}{\#\ left\ prototypes^{(c)} + \epsilon} \times original\ score^{(c)}.$$

We actually implemented this adjustment, yet encountered a new issue: what if all six prototypes are derived from the same training image? In such cases, the final score will trend towards either $+\infty$ or $-\infty$, depending on the value of ϵ. While this may not significantly impact the training image itself, as the predicted score for the correct class becomes extremely high, it can adversely affect other images in the batch, resulting in disproportionately large scores.

Unfortunately, we were unable to modify this implementation due to time constraints, but we do have potential solutions for this issue. One approach is to retain one prototype when all prototypes derive from a single image, rather than masking them all. This adjustment would enhance the completeness of the self-masking mechanism.

References

[1] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS ONE*, 10, 2015. 1

[2] Alina Jade Barnett, Fides Regina Schwartz, Chaofan Tao, Chaofan Chen, Yinhao Ren, Joseph Y. Lo, and Cynthia Rudin. Iaia-bl: A case-based interpretable deep learning model for classification of mass lesions in digital mammography, 2021. 1

[3] Richard Berk. *Machine Learning Risk Assessments in Criminal Justice Settings*. Springer Publishing Company, Incorporated, 1st edition, 2018. 1

[4] Chaofan Chen, Oscar Li, Chaofan Tao, Alina Jade Barnett, Jonathan Su, and Cynthia Rudin. This looks like that: Deep learning for interpretable image recognition, 2019. 1, 2, 3, 4, 5

[5] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks, 2017. 2, 3

[6] Jiankang Deng, Jia Guo, Jing Yang, Niannan Xue, Irene Kotsia, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10):5962–5979, 2022. 2, 3

[7] Jon Donnelly, Alina Jade Barnett, and Chaofan Chen. Deformable protopnet: An interpretable image classifier using deformable prototypes, 2024. 1, 2, 7, 8

[8] Adrian Hoffmann, Claudio Fanconi, Rahul Rade, and Jonas Kohler. This looks like that... does it? shortcomings of latent space prototype interpretability in deep networks, 2021. 2

[9] Qihan Huang, Mengqi Xue, Wenqi Huang, Haofei Zhang, Jie Song, Yongcheng Jing, and Mingli Song. Evaluation and improvement of interpretability for self-explainable part-prototype networks, 2023. 2

[10] Been Kim, Cynthia Rudin, and Julie Shah. The bayesian case model: A generative approach for case-based reasoning and prototype classification, 2015. 1

[11] Oscar Li, Hao Liu, Chaofan Chen, and Cynthia Rudin. Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions, 2017. 1

[12] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition, 2018. 3

[13] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc. 1

[14] Chiyu Ma, Brandon Zhao, Chaofan Chen, and Cynthia Rudin. This looks like those: illuminating prototypical concepts using multiple visualizations. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2024. Curran Associates Inc. 2

[15] Meike Nauta, Annemarie Jutte, Jesper Provoost, and Christin Seifert. *This Looks Like That, Because ... Explaining Prototypes for Interpretable Image Recognition*, page 441–456. Springer International Publishing, 2021. 2

[16] Meike Nauta, Ron van Bree, and Christin Seifert. Neural prototype trees for interpretable fine-grained image recognition, 2021. 1

[17] Anand Nayyar, Lata Gadhavi, and Noor Zaman. Chapter 2 - machine learning in healthcare: review, opportunities and challenges. In *Machine Learning and the Internet of Medical Things in Healthcare*, pages 23–45. Academic Press, 2021. 1

[18] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996. 7

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. 5

[20] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011. 2, 5

[21] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition, 2018. 2, 3

[22] Jiaqi Wang, Huafeng Liu, Xinyue Wang, and Liping Jing. Interpretable image recognition by constructing transparent embedding space. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 875–884, 2021. 2

[23] Thierry Warin and Aleksandar Stojkov. Machine learning in finance: A metadata-based systematic review of the literature. *Journal of Risk and Financial Management*, 14(7), 2021. 1

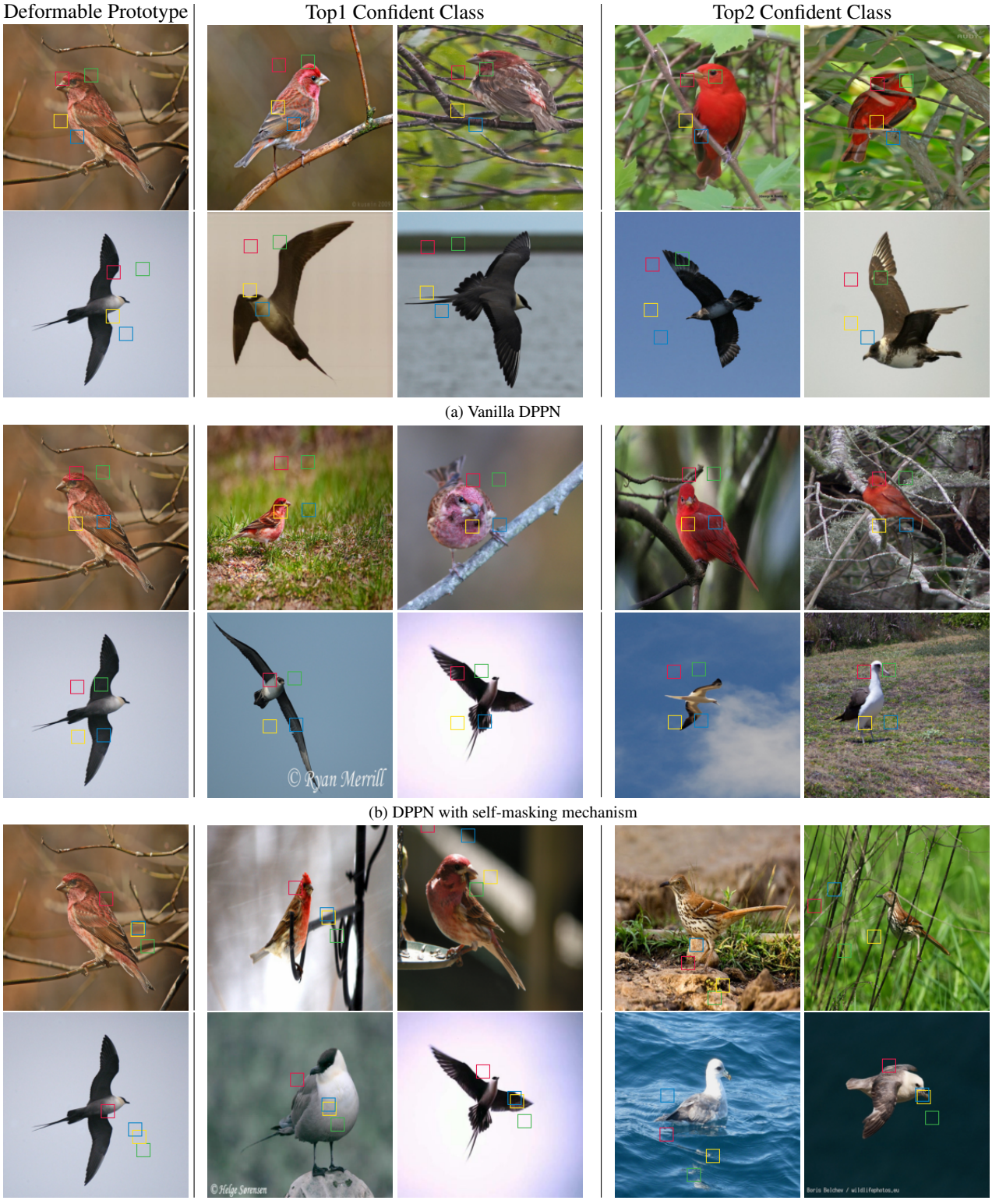
[24] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. 2

[25] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization, 2015. 1

A. Visualization

According to the DPPN paper[7], we utilize local analysis to visualize similarity in CUB-200-2011 dataset of bird images.

Local analysis involves identifying and visualizing the two most similar deformable prototypes (sub-components of the neural network designed to identify key features within images) for each of several test images. These prototypes are organized by the network to match and compare with test images. The similarity is evaluated based on how closely parts of the prototypes correspond to parts of the test images. The most similar prototypes generally belong to the same class as the test image and show semantic correspondence to specific image patches. Some samples are shown in Figure 8.



(a) Vanilla DPPN

(b) DPPN with self-masking mechanism

(c) DPPN with multi-head offset

Figure 8. Local Analysis