Learning Support and Trivial Prototypes for Interpretable Image Classification

Chong Wang\(^1\) Yuyuan Liu\(^1\) Yuanhong Chen \(^1\) Fengbei Liu\(^1\) Yu Tian\(^2\)
Davis McCarthy\(^3\) Helen Frazer\(^4\) Gustavo Carneiro\(^5\)

\(^1\) Australian Institute for Machine Learning, University of Adelaide
\(^2\) Harvard University \(^3\) St Vincent’s Institute of Medical Research
\(^4\) St Vincent’s Hospital Melbourne \(^5\) CVSSP, University of Surrey

Abstract

Prototypical part network (ProtoPNet) methods have been designed to achieve interpretable classification by associating predictions with a set of training prototypes, which we refer to as trivial prototypes because they are trained to lie far from the classification boundary in the feature space. Note that it is possible to make an analogy between ProtoPNet and support vector machine (SVM) given that the classification from both methods relies on computing similarity with a set of training points (i.e., trivial prototypes in ProtoPNet, and support vectors in SVM). However, while trivial prototypes are located far from the classification boundary, support vectors are located close to this boundary, and we argue that this discrepancy with the well-established SVM theory can result in ProtoPNet models with inferior classification accuracy. In this paper, we aim to improve the classification of ProtoPNet with a new method to learn support prototypes that lie near the classification boundary in the feature space, as suggested by the SVM theory. In addition, we target the improvement of classification results with a new model, named ST-ProtoPNet, which exploits our support prototypes and the trivial prototypes to provide more effective classification. Experimental results on CUB-200-2011, Stanford Cars, and Stanford Dogs datasets demonstrate that ST-ProtoPNet achieves state-of-the-art classification accuracy and interpretability results. We also show that the proposed support prototypes tend to be better localised in the object of interest rather than in the background region.

1. Introduction

Deep convolutional neural networks (CNN) \([28, 30, 16]\) have had remarkable achievements in various visual tasks, e.g., image recognition \([16]\) and object detection \([39]\). Despite the excellent feature extraction and discrimination ability, CNNs are generally treated as black-box models due to their complex architectures, high-dimensional feature spaces, and the enormous number of learnable parameters. Such lack of interpretability hinders their successful application in fields that require understandable and transparent decisions \([40]\), e.g., disease diagnosis \([47, 13]\), financial risk assessment \([33]\), and autonomous driving \([23]\).

Recently, increasing attention has been dedicated to the development of interpretable deep-learning models \([25, 1, 4, 3]\). A particularly interesting strategy is the prototype-based gray-box models, e.g., prototypical part network (ProtoPNet) \([4, 11]\). These methods are inherently interpretable since they can explain the model’s decisions by showing image classification activation maps associated with a set of class-specific image prototypes. These prototypes are automatically learned from training samples, with classification score being computed by comparing testing image parts to the learned training prototypes.

ProtoPNet \([4]\) is trained to learn a classifier from a set of class-specific prototypes by minimising the cross-entropy classification loss and two additional regularisation losses, namely: 1) a clustering loss that pulls together training image patches to at least one prototype of its own class; and 2) a separation loss that pushes apart training image...
A combination of these two losses pushes the prototypes as far as possible from the classification boundary, but still within the class distribution, so we call them trivial prototypes, as shown in Fig. 1(a). We also display the trivial prototypes, learned with a feed-forward neural network\(^1\), for the two-moon problem in Fig. 2(a). Notice that these trivial prototypes are located far from the classification boundary. In fine-grained visual classification, the trivial prototypes can mistakenly focus on background regions instead of on the object of interest \([42, 41]\), particularly for those classes with subtle foreground (object) differences but large background variations, as shown in Fig. 3. Different from ProtosNet’s trivial prototypes, the support vector machine (SVM) \([7]\) classifier relies on a set of support vectors that are close to the classification boundary, as in Fig. 2(c). These support vectors are often treated as hard samples. Motivated by SVM, we propose the derivation of support (i.e., hard-to-learn) prototypes for ProtoPNet methods.

In this paper, we propose an alternative learning strategy for ProtoPNet, which forces the learned prototypes to resemble SVM’s support vectors and to be located as close as possible to the classification boundary. The strategy is formulated by a new closeness loss that minimises the distance between prototypes of different classes. As shown in Fig. 1(b), our new loss enforces the prototypes to move closer to the classification boundary, as also demonstrated by Fig. 2(b) revealing that the support prototypes produced by the introduction of our new closeness loss are indeed more similar to the support vectors of SVM in Fig. 2(c). Furthermore, to improve the classification accuracy, we propose a new ST-ProtoPNet method that integrates both the support and trivial prototypes. The ST-ProtoPNet leverages the two distinct and complementary sets of prototypes to capture both hard (i.e., close to the boundary) and easy (i.e., far from the boundary) visual features for classification.

\(^1\)The network has an input layer of 2 nodes, a hidden layer of 256 nodes (activated by tanh), and an output layer of 2 nodes (activated by sigmoid).

Figure 2. Two-moon results. (a) Trivial prototypes and training samples in the feature (top) and data (bottom) spaces from the original ProtoPNet \([4]\). (b) Support prototypes and training samples in the feature (top) and data (bottom) spaces from our method. (c) Support vectors and training samples from a Radial Basis Function (RBF) kernel based SVM \([7]\). In (a) and (b), each prototype is projected onto the nearest training sample in the feature space.

The major contributions of this work are:

1. We provide the first study that makes an analogy between the prototype learning from ProtoPNet and support vector learning from SVM, where we propose support (i.e., hard-to-learn) prototypes that can improve classification accuracy and interpretability.

2. We present a new ST-ProtoPNet method to exploit both support and trivial prototypes for interpretable image classification, where the two sets of prototypes can provide complementary information to improve classification accuracy.

3. We conduct extensive experiments on three benchmarks, showing that our ST-ProtoPNet outperforms current state-of-the-art (SOTA) methods in terms of classification accuracy and interpretability.

In our experiments, we also demonstrate that the trivial and support prototypes have different characteristics, where the trivial prototypes tend to focus on both local parts of the visual object of interest and the background, while the support prototypes mainly focus on object parts belonging to the visual class of interest.

2. Related Work

2.1. Classification Interpretability

The interpretation of classification results produced by deep neural networks can be achieved by a variety of post-hoc explanation techniques, e.g., explanatory surrogates \([34, 56, 44]\), counterfactual examples \([15, 46, 19]\), and saliency visualisation \([45, 55, 59, 43]\). Alternatively, prototype-based interpretable techniques can access the model’s inner computations. ProtoPNet \([4]\) is the original work that uses class-specific prototypes for interpretable image classification. Similar to ProtoPNet, TestNet \([53]\) constructs class-specific transparent basis concepts on Grassmann manifold for the interpretable classification. Derived from ProtoPNet, Deformable ProtoPNet \([11]\) employs spatially-flexible and deformable prototypes to adaptively capture meaningful object features. In ProtoPShare \([42]\), a data-dependent merge-pruning method is presented to share prototypes among classes, which can reduce the number of prototypes used for classification. In contrast, ProtoPool \([41]\) introduces a fully differentiable prototype assignment strategy to reduce the number of prototypes. In
Proto2Proto [20], a knowledge distillation method is designed to transfer interpretability from a teacher ProtoPNet to a shallow student ProtoPNet. ProtoTree [35] integrates the prototype learning into a binary neural decision tree that can explain its predictions by tracing a decision path throughout the tree. ViT-Net [24] further establishes the prototype neural tree structure on visual transformers [12].

Because of the ability to self-explain classification results, prototype-based interpretability (e.g., ProtoPNet) has been widely utilised not only in the computer vision applications above, but also in medical imaging [2, 22, 51] and face recognition [48]. However, an open question faced by these methods is if the prototypes being learned are the ideal ones in terms of classification and interpretability.

2.2. SVM vs Prototype-based Classification

To better understand the role of prototypes, we consider the support vector machine (SVM) [7] classifier that finds support vectors to represent classes. More specifically, SVM learns the maximum-margin classifier defined by a classification boundary that maximises the distance to the closest training samples, which are the support vectors for the classes. The testing of SVM consists of computing a weighted similarity between a testing sample and the support vectors. It is interesting to note that the testing of prototype-based classifiers is also based on measuring the similarity between a testing image and a set of class-specific prototypes learned from the training process. Although the testing of SVM and prototype-based classifiers are similar, their training procedures are different. First, the training of a prototype-based classifier learns a fixed number of prototypes [4, 11], while the SVM classifier learns to weight a variable number of support vectors from the training set. Second, in prototype-based classifiers, the learned prototypes tend to be far from the classification boundary, which is contrary to the SVM training objective mentioned above.

The study of deep learning methods from an SVM theoretical perspective is a rich area of research [9, 38, 6], but is contrary to the SVM training objective mentioned above. The study of deep learning methods from an SVM theoretical perspective is a rich area of research [9, 38, 6], but is contrary to the SVM training objective mentioned above.

2.3. Interpretable Ensemble Classification

Ensemble classification [10] is a classical machine learning approach that combines the results from multiple classifiers, with the goals of improving learning generalisation and classification calibration. The use of interpretable ensemble strategies has been explored in [4, 53, 35, 11, 41], which is achieved by summing the classification logits of multiple prototype-based classifiers (e.g., ProtoPNets trained with different CNN backbones). In this work, we propose an interpretable ensemble classification by combining the predictions of two ProtoPNets with highly distinctive prototypes (i.e., support and trivial prototypes), which is different from previous studies where the type of prototypes produced by each classifier is very similar given that the same training objective is used for each classifier.

3. Preliminaries

We assume to have a training set \(D = \{(x_n, y_n)\}_{n=1}^{|D|}\), where \(x \in \mathcal{X} \subset \mathbb{R}^{H \times W \times R}\) is an image with \(R\) colour channels and \(y \in \mathcal{Y} \subset \{0, 1\}^C\) is a one-hot vector representation of the image class label. The interpretable ProtoPNet [4, 11] is trained to learn a set of prototypes \(P = \{p_m\}_{m=1}^M\), where \(p_m \in \mathbb{R}^{R_{1} \times R_{2} \times D}\), with each of the \(C\) classes containing \(M / C\) prototypes. Without loss of generality, we assume \(p_1 = p_2 = 1\), but the extension to general values is trivial. A typical ProtoPNet comprises four components: a CNN backbone, add-on layers, a prototype layer, and a fully connected (FC) layer. An input image \(x\) is fed to the CNN backbone \(f_\theta : \mathcal{X} \rightarrow \mathcal{F}\) (parameterised by \(\theta \in \Theta\), where \(\mathcal{F} \subset \mathbb{R}^{H \times W \times D}\)) and then passed on to the add-on layers, denoted by \(f_\omega : \mathcal{F} \rightarrow \mathcal{V}\) (parameterised by \(\omega \in \Omega\)), to produce a feature map \(V \in \mathcal{V} \subset \mathbb{R}^{H \times W \times D}\). The prototype layer computes the similarity between the feature map \(V\) and the \(M\) \(D\)-dimensional prototypes \(\{p_m\}_{m=1}^M\) to generate \(M\) similarity maps \(S_{m}^{(i,j)} = \text{sim}(\phi(i,j,:), p_m), \) where \(i \in \{1, ..., \bar{H}\}, j \in \{1, ..., \bar{W}\}\), and \(\text{sim}(\cdot, \cdot)\) represents a similarity measure, e.g., cosine similarity [11] and projection metric [53]. The prototype layer outputs \(M\) similarity scores from max-pooling \(S = \left\{ \max_{i \in \{1, ..., \bar{H}\}, j \in \{1, ..., \bar{W}\}} S_{m}^{(i,j)} \right\}_{m=1}^M\), which are fed to the FC layer \(f_\phi : S \rightarrow \Delta\), parameterised by \(\phi \in \Phi\), to produce the classification prediction \(\hat{y} \in \Delta \subset \{0, 1\}^C\), where \(\Delta\) denotes the probability space for \(C\) classes.

4. ST-ProtoPNet

An overview of our proposed ST-ProtoPNet method is illustrated in Fig. 4, which comprises a shared CNN backbone \(f_\theta(\cdot)\), two interpretable ProtoPNet classification branches, namely: 1) the support ProtoPNet represented by add-on layers \(f_{\omega_{s}(\cdot)}(\cdot)\), prototype layer with support prototypes \(P_{s}(\cdot)\), and FC layer \(f_{\phi_{s}(\cdot)}(\cdot)\) which outputs the classification probability distribution \(\hat{y}^{(s)}(\cdot) \in \Delta;\) and 2) the trivial ProtoPNet branch with its add-on layers \(f_{\omega_{t}(\cdot)}(\cdot)\), trivial prototypes \(P_{t}(\cdot)\), and FC layer \(f_{\phi_{t}(\cdot)}(\cdot)\) that generates probability predictions \(\hat{y}^{(t)}(\cdot) \in \Delta\). The final classification is obtained by combining the classification logits from both the support and trivial ProtoPNets. In our implementation, we construct the support and trivial ProtoPNets mainly based on the orig-
4.1. Support ProtoPNet

The support ProtoPNet is designed to produce support prototypes that are as close as possible to the classification boundary, as shown in Fig. 1(b) and 2(b). The loss function to optimise the support ProtoPNet branch is defined as:

$$\theta^*, \omega^*(s), \mathcal{P}^*(s), \phi^*(s) = \arg \min_{\theta, \omega, \mathcal{P}, \phi} \sum_{(x,y) \in D} \ell_{sp}(x, y, \theta, \omega(s), \mathcal{P}(s), \phi(s)).$$  \hspace{1cm} (1)

The loss for each training sample \((x, y) \in D\) in Eq. (1) above is represented by:

$$\ell_{sp}(x, y, \theta, \omega(s), \mathcal{P}(s), \phi(s)) = \ell_{ce}(x, y, \theta, \omega(s), \mathcal{P}(s), \phi(s)) - \lambda_1 \ell_{cl}(x, y, \theta, \omega(s), \mathcal{P}(s)) + \lambda_2 \ell_{sp}(x, y, \theta, \omega(s), \mathcal{P}(s)) - \lambda_3 \ell_{cls} \left( \mathcal{P}^{(s)} \right) + \lambda_4 \ell_{ort} \left( \mathcal{P}^{(s)} \right),$$  \hspace{1cm} (2)

where \(\lambda_1, \lambda_2, \lambda_3, \text{ and } \lambda_4\) are hyper-parameters to balance each term, \(\ell_{ce}(\cdot)\) denotes the cross-entropy classification loss, \(\ell_{cl}(\cdot)\) and \(\ell_{sp}(\cdot)\) represent the clustering and separation losses, respectively, which are introduced to regularise the ProtoPNet’s training, as follows:

$$\ell_{cl}(x, y, \theta, \omega(s), \mathcal{P}(s)) = \max_{P \in \mathcal{P}^{(s)}(y)} \max_{v \in \mathbf{V}^{(s)}} \text{sim}(v, p),$$  \hspace{1cm} (3)

$$\ell_{sp}(x, y, \theta, \omega(s), \mathcal{P}(s)) = \max_{P \notin \mathcal{P}^{(s)}(y)} \max_{v \in \mathbf{V}^{(s)}} \text{sim}(v, p),$$  \hspace{1cm} (4)

where \(\mathbf{V}^{(s)} = f_{\omega}(f_{\theta}(x))\) is the feature map extracted from the input image \(x\), \(v\) represents one of the \(H \times W\) feature vectors in \(\mathbf{V}^{(s)}\) obtained by matrix vectorisation, \(p\) is a normalised prototype (i.e., unit vector) in \(\mathcal{P}^{(s)}\), \(\text{sim}(\cdot, \cdot)\)
is one of the similarity functions defined in Sec. 3, and \(\mathcal{P}^{(s)}_y\) denotes the set of prototypes of class \(y\). The clustering loss in Eq. (3) and separation loss in Eq. (4) aim to learn a meaningful feature space in which the image features of a certain class are clustered around the prototypes of the class, and also well separated from those of other classes.

As mentioned in Sec. 1, the effect of the clustering and separation losses above tend to push the prototypes of different classes as far as possible from the classification boundary, resulting in trivial prototypes, as displayed in Fig. 1(a) and Fig. 2(a). In order to learn the proposed support prototypes, we introduce the following novel closeness loss \(\ell_{cls}\) to explicitly enforce the prototypes of different classes to be close to each other, which is formulated as:

$$\ell_{cls}(\mathcal{P}^{(s)}) = \sum_{c_1=1}^{C-1} \sum_{c_2=c_1+1}^{C} \min_{p_m \in \mathcal{P}_{c_1}, p_n \in \mathcal{P}_{c_2}} \| p_m - p_n \|_F^2.$$  \hspace{1cm} (5)

During training, this closeness loss \(\ell_{cls}\) maximises the pair-wise prototype similarity, in the form of dot product between different prototypes in Eq. (5) above, with the goal of pulling the prototypes close to the classification boundary. On the one hand, as the prototypes move gradually towards the classification boundary, they are able to capture harder visual features from training samples. On the other hand, since the prototypes are located near the classification boundary, they can put pressure on the support ProtoPNet’s feature learning (i.e., enforce it to learn highly discriminative feature representations for accurate classification), which is beneficial to extract more meaningful semantic information from training samples.

Ideally, each prototype of a class should focus on unique object parts of the training images (e.g., head, tail, and claw of birds), so that the prototypes can represent rich and diverse visual patterns. However, there is no particular constraints to guarantee such prototype diversity and the issue of prototype duplication [11] often occurs in the ProtoPNet family of models. To encourage the intra-class prototype diversity, we employ an orthonormality loss [53] so that prototypes within a class can represent dissimilar visual patterns of training samples, which is defined as:

$$\ell_{ort}(\mathcal{P}^{(s)}) = \sum_{c=1}^{C} \| P_c^T P_c - I_{M/C} \|_F^2,$$  \hspace{1cm} (6)

where \(\| \cdot \|_F^2\) represents Frobenius norm, \(P_c \in D \times \mathbb{R}^{(M/C)}\) stands for a matrix composed of the prototypes of class \(c\) (prototypes in each column of \(P_c\) are normalised), and \(I_{M/C}\) is an identity matrix of size \(M/C \times M/C\).

4.2. Trivial ProtoPNet

As described in Sec. 4.1, the support ProtoPNet is developed to learn support (i.e., hard-to-learn) prototypes by...
forcing them to be close to the classification boundary. Considering that training samples contain not only hard visual features but also important easy ones that the support prototypes cannot completely capture, we propose to also learn trivial prototypes to provide complementary classification information, and exploit both the support and trivial prototypes for improved interpretable classification.

The loss objective to optimise the trivial ProtoPNet branch is defined as follows:

\[
\arg \min_{\theta, \omega(t), \mathcal{P}(t), \phi(t)} \sum_{(x, y) \in \mathcal{D}} \ell_{trv}(x, y, \theta, \omega(t), \mathcal{P}(t), \phi(t)).
\]

(7)

The loss for each training image \( (x, y) \in \mathcal{D} \) in Eq. (7) above is represented by:

\[
\ell_{trv}(x, y, \theta, \omega(t), \mathcal{P}(t), \phi(t)) = \ell_{ce}(x, y, \theta, \omega(t), \mathcal{P}(t), \phi(t)) - \lambda_1 \ell_{ct}(x, y, \theta, \omega(t), \mathcal{P}(t)) + \lambda_2 \ell_{sp}(x, y, \theta, \omega(t), \mathcal{P}(t)) + \lambda_3 \ell_{dsc}(\mathcal{P}(t)) + \lambda_4 \ell_{ort}(\mathcal{P}(t)),
\]

(8)

where \( \lambda_1, \lambda_2, \lambda_3, \) and \( \lambda_4 \) are hyper-parameters, \( \ell_{ce}() \) is the cross-entropy loss, the clustering loss \( \ell_{ct} \), separation loss \( \ell_{sp} \), and orthornormality loss \( \ell_{ort} \) are the same as in the support ProtoPNet defined in Eq. (3), (4) and (6), respectively.

The trivial ProtoPNet targets the learning of easy prototypes that are far from the classification boundary and have a good discrimination ability. To help achieve this, we introduce a new discrimination loss \( \ell_{dsc} \) to facilitate the inter-class separability between prototypes of different classes. This is formulated by minimising the pair-wise prototype similarities of different classes, as follows:

\[
\ell_{dsc}(\mathcal{P}(t)) = \sum_{c_1 = 1}^{C-1} \sum_{c_2 = c_1 + 1}^{C} \max_{p_m \in P_{c_1}, p_n \in P_{c_2}} \bar{W}^T p_m p_n.
\]

(9)

4.3. Training and Testing

**Training.** Following the training strategies in [4, 11], the training procedure of our ST-ProtoPNet consists of 3 stages: 1) optimisation of the CNN backbone, add-on layers, and prototype layer, using a fixed FC layer initialised with +1.0 and -0.5 for correct and incorrect connection weights, respectively. A warm-up of 5 epochs is involved in this stage by updating only the parameters of add-on layers and prototype layer, with a frozen pre-trained CNN backbone. 2) prototype projection by updating each prototype with its nearest latent training image patch; and 3) optimisation of the FC layer, with an additional \( \ell_1 \) regularisation on the incorrect connection weights (initially fixed at -0.5). In each stage, we alternate the optimisation of each branch of the ST-ProtoPNet between mini-batches. Notice that our method only brings marginal extra model parameters, computational complexity, and training time since the CNN backbone is shared and optimised by both branches.

**Testing.** To exploit the complementary results from both branches of ST-ProtoPNet, its final classification is obtained from the summed logits predicted by the two branches. It is worth noticing that this ensemble strategy introduces no loss of interpretability but improved accuracy.

5. Experiments

We perform experiments on three fine-grained classification benchmark datasets: CUB-200-2011 [50], Stanford Cars [27], and Stanford Dogs [21]. To achieve fair comparison, we follow previous studies [4, 53] by applying offline data augmentations (e.g., random rotation, skew, shear, and left-right flip) on the cropped CUB and cropped Cars datasets (using the bounding boxes provided). We also validate our method on the full (i.e., uncropped) CUB and Dogs datasets, and employ the same online data augmentation methods (e.g., random affine transformation and left-right flip) as used in Deformable ProtoPNet [11]. All images are resized to 224 \( \times \) 224 pixels as network input.

5.1. Experimental Settings

The proposed ST-ProtoPNet method is evaluated on the following CNN architectures: VGG-16, VGG-19, ResNet-34, ResNet-50, ResNet-152, DenseNet-121, and DenseNet-161. All CNN backbones are pre-trained on ImageNet [8], except for ResNet-50, which is pre-trained on tNaturalist [49] for the experiment on full CUB [11]. The add-on layers include two 1 \( \times \) 1 convolutional layers. For simplicity, we utilise the same prototype dimension \( D = 64 \) for all CNN backbones on the three datasets. For cropped CUB and Cars datasets, following [53], we use 10 prototypes (5 support and 5 trivial) per class and the projection metric in the similarity function \( \text{sim}(\cdot, \cdot) \). In full CUB and Dogs datasets, to ensure comparison fairness with Deformable ProtoPNet [11] that uses 10 \( 2 \times 2 \) (full CUB) and 10 \( 3 \times 3 \) (full Dogs) deformable prototypes per class, we utilise the same total number of prototypes, i.e., 40 \( 1 \times 1 \) (20 support and 20 trivial) for full CUB and 90 \( 1 \times 1 \) (45 support and 45 trivial) for full Dogs. Also, we employ the cosine similarity in \( \text{sim}(\cdot, \cdot) \) and obtain \( 14 \times 14 (H = W = 14) \) feature maps by upsampling the original \( 7 \times 7 \) feature maps via a bi-linear interpolation step, as in [11]. Following previous prototype-based methods [4, 53, 11], we set \( \lambda_1 = 0.8, \lambda_2 = 0.48 \) and 0.08 for the support and trivial ProtoPNet branches respectively, \( \lambda_4 = 0.001 \). We choose \( \lambda_3 = 1.0 \) with an ablation provided in the supplementary material.
5.2. Interpretability Evaluation

Rather than performing user-based evaluations [60, 41], whose results are often subjective and difficult to reproduce [54], we leverage the annotated object masks to measure interpretability based on the following metrics:

**Content Heatmap (CH)** [37]: quantifies the percentage of an activation heatmap that lies within the annotated mask. Hence, we expect this metric to be high.

**Outside-Inside Relevance Ratio (OIRR)** [29]: calculates the ratio of mean activation outside the object to mean activation inside the object. A low OIRR indicates a method relies more on the object region and less on the context to support its decision, we thus anticipate OIRR to be low.

**Intersection over Union (IoU)** [5]: measures the mean IoU score, where a threshold of 0.5 is applied on the min-max normalised heatmap to select foreground objects.

**Deletion AUC (DAUC)** [36]: estimates a decrease in the probability of the predicted class as more and more important/activated pixels are removed. The area under the probability curve is defined as DAUC. Consequently, a sharp drop and a low DAUC mean better interpretation.

**Average Inter-class Prototype Distance (AIPD), Average Inter-class Feature Distance (AIFD)**: we had earlier stated that support prototypes of different classes should be close to each other and trivial prototypes should be far from each other. Thus, we compute the average inter-class cosine distance for prototypes and their nearest local feature representations, respectively. We expect AIPD < AIFD for support ProtoPNet and AIPD > AIFD for trivial ProtoPNet.

5.3. Classification Performance

Table 1 presents the classification accuracy (across 5 runs) of our proposed ST-ProtoPNet on cropped CUB and cropped Dogs, where the Baseline is represented by non-interpretable black-box CNN models. As can be seen, our ST-ProtoPNet outperforms other competing methods across all backbones for the task of bird species classification. Also, our method achieves the best results for the car model identification task when using VGG and DenseNet architectures as the CNN backbone. In particular, our VGG19-based ST-ProtoPNet reaches an average accuracy of 83.2% and 91.7% on CUB and Cars, respectively, surpassing other methods with the most improvements across all backbones. Moreover, the support ProtoPNet generally performs better than methods utilising only trivial prototypes (e.g., ProtoPNet, TesNet, and Trivial ProtoPNet), showing the importance of learning support prototypes for the interpretable classification. It is worth noting that our ST-ProtoPNet produces superior performance over the support ProtoPNet method, indicating that both support and trivial prototypes are useful and can provide complementary information for achieving accurate and interpretable classification.

Table 2 shows the classification results on full CUB and full Dogs. In both datasets, the classification accuracy of the original ProtoPNet method is generally worse than the non-interpretable counterpart (Baseline) for many CNN backbones. On the other hand, the accuracy by the trivial ProtoPNet and support ProtoPNet are substantially better than those by Baseline, ProtoPNet, and Deformable ProtoPNet. However, our ST-ProtoPNet achieves more significant performance gains and exhibits the best accuracy across most backbones, particularly when using a large number of prototypes (i.e., 40 $1 \times 1$ prototypes per class for CUB and 90 $1 \times 1$ prototypes per class for Dogs), demonstrating the effectiveness of utilising both the trivial and support prototypes for the interpretable image classification. Additionally, when using a smaller number of prototypes, i.e., 10 $1 \times 1$ prototypes per class, our ST-ProtoPNet still has competitive classification accuracy across multiple backbones.

We further compare our ST-ProtoPNet with other deep-learning methods that can provide different levels of interpretability on CUB, with results shown in Table 3, where * and ** denote ensemble of models with different backbones. As evident, an ensemble of three ST-ProtoPNNets can achieve high accuracy (87.9% on cropped images, 88.2% on full images), outperforming competing methods that are also based on an ensemble of three models (e.g., ProtoTree, TesNet, and ProtoPool). Moreover, the ensemble of five ST-ProtoPNNets outperforms all other competing methods and obtains the best classification accuracy of 88.1% and 88.4% on cropped and full CUB images, respectively. More results on Cars and Dogs are given in the supplementary material.

5.4. Interpretability Comparison

We assess the model interpretability on full CUB using the annotated bird segmentation mask\(^2\). Quantitative results on the test set are given in Table 4, where all methods are based on the VGG19 backbone. We use GradCAM [43] for the non-interpretable baseline. For prototype-based methods, we average the activation map of all prototypes of a class to compute the metrics. We can see our proposed support ProtoPNet can effectively improve the interpretability in all measures, showing the interpretations produced by our support prototypes are more likely to be object-dependent and focus less on context cues. Also, our ST-ProtoPNet method shows better interpretability results than the support ProtoPNet in terms of CH, OIRR, and DAUC. Compared with the original ProtoPNet [4], our ST-ProtoPNet obtains significant interpretability improvements. We show some example activation maps in the supplementary material, and an experiment with prototype pruning is also provided.

Table 5 presents the computed AIPD and AIFD for the support and trivial ProtoPNet on cropped CUB, using VGG19 and ResNet34 as CNN backbones. As evident, the AIPD is indeed smaller than AIFD for the support ProtoPNet.

\(^2\)http://www.vision.caltech.edu/datasets/
Table 1. Classification accuracy (%) on cropped CUB-200-2011 and Stanford Cars by competing methods using different CNN backbones.

<table>
<thead>
<tr>
<th>Method</th>
<th>VGG16</th>
<th>VGG19</th>
<th>ResNet34</th>
<th>ResNet50</th>
<th>Dense121</th>
<th>Dense161</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>73.1 ± 0.4</td>
<td>74.7 ± 0.4</td>
<td>82.2 ± 0.3</td>
<td>80.8 ± 0.4</td>
<td>81.8 ± 0.1</td>
<td>82.1 ± 0.2</td>
</tr>
<tr>
<td>ProtoPNet [4]</td>
<td>77.2 ± 0.2</td>
<td>77.6 ± 0.2</td>
<td>78.6 ± 0.1</td>
<td>79.2 ± 0.3</td>
<td>79.0 ± 0.2</td>
<td>80.8 ± 0.3</td>
</tr>
<tr>
<td>TesNet [53]</td>
<td>81.3 ± 0.2</td>
<td>81.4 ± 0.1</td>
<td>82.8 ± 0.1</td>
<td>82.7 ± 0.2</td>
<td>84.8 ± 0.2</td>
<td>84.6 ± 0.3</td>
</tr>
<tr>
<td>Trivial ProtoPNet</td>
<td>80.8 ± 0.2</td>
<td>81.2 ± 0.2</td>
<td>82.5 ± 0.2</td>
<td>83.1 ± 0.3</td>
<td>83.9 ± 0.3</td>
<td>84.6 ± 0.3</td>
</tr>
<tr>
<td>Support ProtoPNet</td>
<td>81.7 ± 0.2</td>
<td>81.8 ± 0.3</td>
<td>83.0 ± 0.1</td>
<td>83.6 ± 0.2</td>
<td>84.7 ± 0.2</td>
<td>85.2 ± 0.3</td>
</tr>
<tr>
<td>ST-ProtoPNet (ours)</td>
<td>82.9 ± 0.2</td>
<td>83.2 ± 0.2</td>
<td>83.5 ± 0.1</td>
<td>84.1 ± 0.2</td>
<td>85.4 ± 0.2</td>
<td>86.1 ± 0.2</td>
</tr>
</tbody>
</table>

Table 2. Classification accuracy (%) on full CUB-200-2011 and Stanford Dogs datasets by competing approaches using different CNN backbones, where \( p_1 \times p_2 \) denotes the spatial shape of prototypes and \( k \) represents \( k \) prototypes per class.

<table>
<thead>
<tr>
<th>Interpretability level</th>
<th>Method</th>
<th>CUB</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object-level attention</td>
<td>Baseline</td>
<td>70.9</td>
<td>81.5 (b) 84.1 (f)</td>
</tr>
<tr>
<td></td>
<td>ProtoPNet [4]</td>
<td>70.3</td>
<td>82.0 (b) 84.5 (f)</td>
</tr>
<tr>
<td></td>
<td>TesNet [53]</td>
<td>75.8</td>
<td>86.5 (b) 89.4 (f)</td>
</tr>
<tr>
<td></td>
<td>Deformable ProtoPNet [11]</td>
<td>77.6</td>
<td>87.3 (b) 89.7 (f)</td>
</tr>
<tr>
<td></td>
<td>Trivial ProtoPNet</td>
<td>80.0</td>
<td>87.2 (b) 88.1 (f)</td>
</tr>
<tr>
<td></td>
<td>Support ProtoPNet</td>
<td>80.4</td>
<td>87.5 (b) 88.2 (f)</td>
</tr>
<tr>
<td></td>
<td>ST-ProtoPNet (ours)</td>
<td>76.8</td>
<td>86.6 (b) 88.2 (f)</td>
</tr>
<tr>
<td></td>
<td>ST-ProtoPNet (ours)</td>
<td>81.0</td>
<td>88.0 (b) 88.2 (f)</td>
</tr>
</tbody>
</table>

Table 3. Classification accuracy and interpretability level of different methods on CUB-200-2011. “b” and “f” denote the model is trained and tested on cropped and full images, respectively. *: Ensemble of three models, **: Ensemble of five models.

Table 4. Quantitative interpretability results on full CUB test set. DefProto = Deformable ProtoPNet, TrvProto = Trivial ProtoPNet, SptProto = Support ProtoPNet, ST-Proto = ST-ProtoPNet.

Net while AIPD is larger than AIFD for the trivial ProtoPNet. This result indicates that the support prototypes of different classes lie closer than their local feature representations and are more inclined to focus on visually similar (i.e., hard-to-learn) object parts of different classes.

5.5. Visualisation Analysis

To explore the differences between the support and trivial prototypes, we select 5 categories of birds with visually similar features from cropped CUB to train the support and trivial ProtoPNet methods, with the learned prototypes shown in Fig. 5. We notice the support prototypes can cap-
Figure 6. T-SNE results of support and trivial prototypes. The prototypes and their nearest latent training features are marked with stars and dots, respectively. We show results before the prototype projection stage to better visualise the relation between the prototypes and features. Each colour represents a different class.

5.6. Ablation Study

As evident, each ProtoPNNet branch calculates its own classification logits (weighted sum of similarity scores), which is then combined to generate the final prediction. Specifically, when classifying a Parakeet Auklet, the support prototypes are quite active on the bird’s beak and belly. Meanwhile, the trivial prototypes have high activations on the bird’s lower surface and neck. In this case, the support ProtosNet may treat the background as an easy pattern to learn, focusing less on the object’s visual parts of the class. In Fig. 6, we show the t-SNE result from the 5-category bird classification, where we note the support prototypes of different classes are located closer to each other, in comparison with the trivial prototypes.

Fig. 7 shows an example of the interpretable reasoning for our ST-ProtoPNNet in classifying a testing bird image. As evident, each ProtoPNNet branch calculates its own classification logits (weighted sum of similarity scores), which is then combined to generate the final prediction. Specifically, when classifying a Parakeet Auklet, the support prototypes are quite active on the bird’s beak and belly. Meanwhile, the trivial prototypes have high activations on the bird’s lower surface and neck. In this case, the support ProtosNet obtains a relatively higher similarity score (22.925), compared with the trivial branch (20.313). Note that our ST-ProtoPNNet exploits both the support and trivial prototypes to capture much richer representations of the object.

Table 6. Ablation analysis of the closeness loss $\ell_{ct}$ in Eq. (5) to learn support prototypes, and discrimination loss $\ell_{disc}$ in Eq. (9) to learn trivial prototypes on full CUB-200-2011 and Stanford Dogs.

<table>
<thead>
<tr>
<th>Method</th>
<th>$\ell_{ct}$</th>
<th>$\ell_{sp}$</th>
<th>$\ell_{act}$</th>
<th>$\ell_{desc}$</th>
<th>$\ell_{disc}$</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>86.5</td>
</tr>
<tr>
<td>Trivial ProtoPNNet</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>87.2</td>
</tr>
<tr>
<td>Support ProtoPNNet</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>87.5</td>
</tr>
<tr>
<td>ST-ProtoPNNet (ours)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>88.0</td>
</tr>
</tbody>
</table>

Combining Support and Trivial Prototypes. We also investigate the importance of integrating the two complementary sets of support and trivial prototypes for improved classification. To achieve this, we first train a two-branch model where both branches learn the same type of prototypes and the final result is produced by the ensemble of them (Trivial Ensemble and Support Ensemble). Besides, for our ST-ProtoPNNet, we also provide results of its individual branches (Trivial Branch and Support Branch). Table 7 shows the experimental results on cropped CUB. We can notice that combining the two different types of prototypes (ST-ProtoPNNet) achieves superior performance over combining only the same type of prototypes (Trivial Ensemble and Support Ensemble), indicating that our performance improvements are from not only the ensemble strategy but also the two complementary sets of prototypes. Also, ST-ProtoPNNet indeed exhibits higher accuracy than its individual branches, further verifying that the results from the two branches are complementary, and the combination of them is effective to improve the final classification accuracy.

6. Conclusion and Future Work

In this paper, we proposed the ST-ProtoPNNet to exploit both support (i.e., hard-to-learn) and trivial (i.e., easy-to-learn) prototypes, where the two sets of prototypes can pro-
vide complementary results for the interpretable image classification. Our ST-ProtoPNet is a general approach that can be easily applied to existing prototype-based interpretable models. One limitation for our method is that we empirically adopt the same number of support and trivial prototypes and the same total number of prototypes for each class. Considering the different learning difficulties and imbalanced training samples among classes in other real-world datasets, e.g., ImageNet [8], a better way to adaptively learn a flexible number of support and trivial prototypes is needed and deserves to be further investigated in our future work. Moreover, given that we mimic the behaviour of the support vectors of SVM classifier to obtain the support prototypes by forcing them to be as close as possible to the classification boundary, we plan to develop new methods to learn prototypes with gradient-based kernel techniques, e.g., neural tangent kernel [18] and path kernel [9].

Acknowledgements. This work was supported by funding from the Australian Government under the Medical Research Future Fund - Grant MRFAI000090 for the Transforming Breast Cancer Screening with Artificial Intelligence (BRAx) Project, and the Australian Research Council through grant FT190100525.

References


