Pixel-Grounded Prototypical Part Networks

Supplemental Material

Zachariah Carmichael[1,2] Suhas Lohit[2] Anoop Cherian[2] Michael J. Jones[2] Walter J. Scheirer[1]
[1]University of Notre Dame
[2]Mitsubishi Electric Research Laboratories
zcarmich@nd.edu, {slohit,anoop.cherian,mjones}@merl.com, walter.scheirer@nd.edu

Contents

A. Symbols and Functions

Tables A1 and A2 describe the functions and symbols used in this paper, respectively.

B. More Results

Figures B1 and B2 present the discovered "Goldilocks" zone backbones evaluated in PixPNet for CUB-200-2011 and Stanford Cars, respectively. As can be seen, the Pareto front is mostly retained, demonstrating that the ImageNette approach is a good proxy for backbone selection for PixPNet. The results for various ImageNet-pre-trained ProtoPartNNs are provided in Table B3. Figures B5 and B6 shows more examples of explanations on CUB-200-2011 and Stanford Cars, respectively. We also include a video with the supplemental material submission that demonstrates the relevance ordering test for the CUB-200-2011 dataset.

Function	Description
φ	The distance function
f_{core}	The core neural network backbone
f_{add}	The add-on layers to the backbone
f	The feature encoding function
g	The prototype layer
h	The readout layer
v	The similarity function
π	The similarity map function
`patches`	Yields patches from an embedded image
$\mathcal{L}_{\text{total}}$	The total loss function
$\mathcal{L}_{\text{xent}}$	The cross-entropy loss function
$\mathcal{L}_{\text{cls}}$	The cluster loss function
$\mathcal{L}_{\text{sep}}$	The separation loss function
$\mathcal{L}_h$	The readout loss function

Table A1. Functions used in this paper.

C. Additional Receptive Field Algorithm Details

Here, we provide addition description of are function receptive field computation algorithm, `FunctionalRF`. Some complexity of the algorithm comes from handling arbitrary non-sequential architectures. `FunctionalRF`, takes a neural network as input and outputs the *exact* receptive field of every neuron in the neural network. Recall that a neuron is a function of a *subset* of pixels defined by its receptive field. `FunctionalRF` represents receptive fields as hypercubes (multidimensional tensor slices). For convolutional neural networks, we have four dimensions, but the batch size can safely be ignored. We use the notation $[\![a, b]\!]$ to denote the slice (discrete interval) between a and b. The computation is outlined (see Algorithm 1) for image data for simplicity – the algorithm works for any number of dimensions or type of data. Given the directed acyclic

Symbol	Shape	Description
D	–	The prototype dimensionality
P	–	The number of prototypes
N	–	The number of samples
C	–	The number of classes
H	–	The height of a sample
W	–	The width of a sample
H_z	–	The height of $\boldsymbol{z}$
W_z	–	The width of $\boldsymbol{z}$
H_p	–	The height of a prototype
W_p	–	The width of a prototype
$\boldsymbol{p}$	$D \times H_p \times W_p$	A prototype
$\boldsymbol{P}$	$P \times D \times H_p \times W_p$	The tensor of all prototypes
$\boldsymbol{z}$	$D \times H_p \times W_p$	A single patch (embedded vector)
$\boldsymbol{Z}$	$D \times H_z \times W_z$	A tensor of embeddings
$\boldsymbol{s}$	P	A set of similarity scores
$\boldsymbol{S}$	$H_z/H_p \times W_z/W_p$	A similarity map for one prototype
$\mathbf{D}$	$N \times 3 \times H \times W$	A dataset
$\mathbf{X}$	$N \times 3 \times H \times W$	All samples in a dataset
$\boldsymbol{x}$	$3 \times H \times W$	A sample
$\boldsymbol{Y}$	–	All ground truth labels
y	–	The ground truth label
$\hat{\boldsymbol{y}}$	C	The predicted logits
$\hat{y}$	–	The prediction
λ_{cls}	–	Loss function coefficient for $\mathcal{L}_{\text{cls}}$
λ_{sep}	–	Loss function coefficient for $\mathcal{L}_{\text{sep}}$
λ_h	–	Loss function coefficient for $\mathcal{L}_h$
$\boldsymbol{W}_h$	$P \times C$	The weight matrix of h
w_h	–	An element of $\boldsymbol{W}_h$

Table A2. Symbols used in this paper.

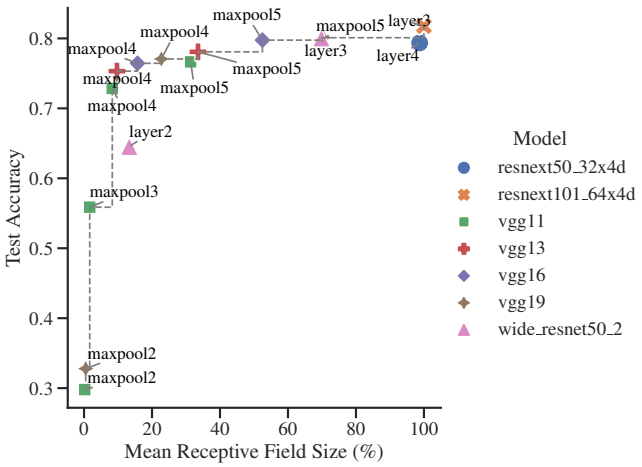

Figure B1. Our approach, PIXPNET, evaluation on CUB-200-2011 with various backbones discovered by the ImageNette proxy approach outlined in the main text. The Pareto front is given by the dashed line.

computation graph of a neural network $\mathcal{G}$, we can traverse the topologically sorted graph node by node to satisfy input dependencies. At the start, we initialize the receptive field (`rf` attribute) of the input node $v_0 \in \mathcal{G}$ as $1 \times 1 \times 1$ slices into the input image. Each consecutive node v_k performs operation-dependent indexing into the receptive fields of its

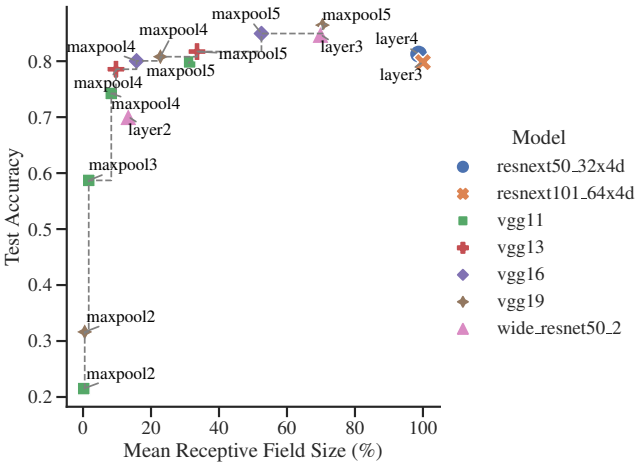

Figure B2. Our approach, PIXPNET, evaluation on Stanford Cars with various backbones discovered by the ImageNette proxy approach outlined in the main text. The Pareto front is given by the dashed line.

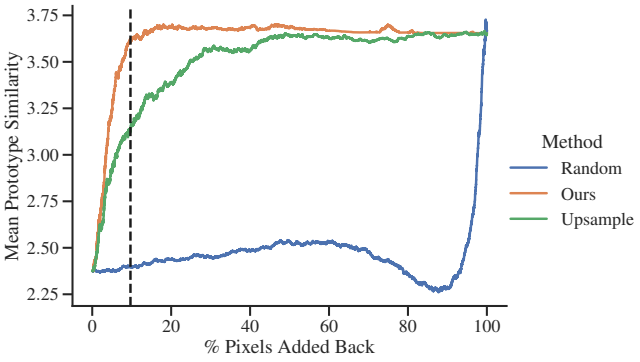

Figure B3. Relative ordering test for a VGG13@maxpool4. The original, random, and our pixel space mappings are compared. We achieve the highest AUSC and the lowest %2R. PRP is not shown because its implementation only supports ResNet models. The black dashed line indicates the mean receptive field of maxpool4.

incoming nodes $\mathcal{U}_k$. This indexing is operation-dependent and is encapsulated by the function `take_from`$(\cdot)$. For instance, the slices for a 2D convolution with a $5{\times}5$ kernel, stride of 1, and c_{in} channels at output position $3, 3$ would be $\{\{[\![1, c_{\text{in}}]\!], [\![1, 5]\!], [\![1, 5]\!]\}\}$ where $[\![a, b]\!]$ denotes the slice between a and b. After, we merge (`merge`$(\cdot)$) as many hypercubes as possible into a larger hypercube (consider, *e.g.*, one hypercube inside of another) to greatly reduce the space and time complexities. Figure C7 gives several examples of this merge operation. Finally, the receptive field-augmented graph $\mathcal{G}$ is returned.

BBox	D1	D2	D3	Model	f	Expl. Size +	Expl. Size ±	P	MRF	Accuracy	±	Code Avail.	Val. Set
				PIXPNET	VGG19@maxpool5	**10**	**10**	2000	70.4	**86.44**	0.2	✓	✓
	✓	✓	✓	PIXPNET	VGG16@maxpool5	**10**	**10**	2000	52.5	84.94	0.1	✓	✓
				PIXPNET	VGG13@maxpool5	**10**	**10**	2000	33.5	81.72	0.2	✓	✓
✗				PIXPNET	VGG16@maxpool4	**10**	**10**	2000	15.8	80.05	0.3	✓	✓
				SUPPORT PROTOPNET [38]	ResNet152	180	35280	17640	100	**87.30**	–	✓	✗
	✓	✗	✓	DEFORMABLE [7]	ResNet152	180	35280	17640	100	86.50	–	✓	✗
				ST-PROTOPNET [38]	ResNet152	20	3920	1960	100	85.30	–	✓	✗
				ST-PROTOPNET [38]	DenseNet161	20	3920	1960	100	<u>92.70</u>	0.2	✓	✗
				TESNET [39]	DenseNet161	20	3920	1960	100	<u>92.60</u>	0.3	✓	✗
				PROTOPOOL [28]	ResNet34	20	390	195	100	89.30	0.1	✓	✗
✓	✓	✗	✓	PROTOPNET [5]	VGG19	20	3920	1960	100	87.40	0.3	✓	✗
				PROTOTREE [23]	ResNet34	22	390	195	100	86.60	0.2	✓	✗
				PROTOPSHARE [29]	ResNet34	960	960	480	100	86.38	–	✓	✗
				PROTO2PROTO [14]	ResNet18	20	3920	1960	100	84.00	–	✓	✗
✗	✗	✗	✗	VIT-NET [16]	SwinT-B	12	126	63	100	**95.00**	–	✓	✗
	✗	✗	✓	PROTOPFORMER [40]	CaiT-XXS-24	30	5880	2940	100	91.04	–	✓	✗

Table B3. ProtoPartNN results on Stanford Cars with ImageNet used for pre-training. Columns D1, D2, and D3 correspond to the three desiderata established in the main text. "BBox" indicates whether a method crops each image using a bounding box annotation. The best results of ProtoPartNNs with and without such annotations are **bold** and <u>underlined</u>, respectively. Best results that are within one standard deviation of each other are all emphasized. The table is split based on whether the method meets at least two desiderata.

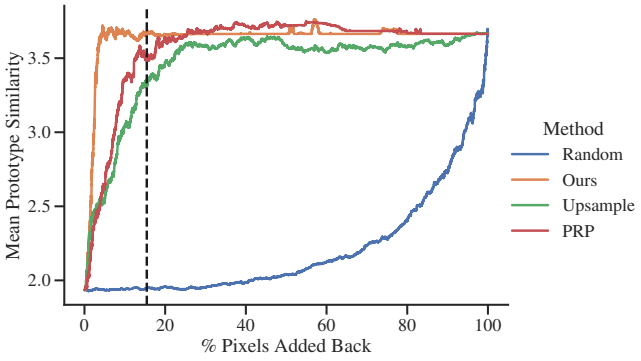

Figure B4. Relative ordering test for a ResNet18@layer2. The original, random, and our pixel space mappings are compared. We achieve the highest AUSC and the lowest %2R. The black dashed line indicates the mean receptive field of layer2.

Algorithm 1: `FunctionalRF`($\mathcal{G}$)

Input: $\mathcal{G}$, the computation graph
Output: $\mathcal{G}$, augmented with receptive field information

Topologically sort $\mathcal{G}$;
`// Initialize each input node rf element as a 1×1×1 hypercube`
$v_0.\texttt{rf}_{dij} \leftarrow \{\{[\![1,3]\!],[\![i,i]\!],[\![j,j]\!]\}\}$;
for $v_k \in \mathcal{G}$ **do**
  $\mathcal{U}_k \leftarrow$ all incoming nodes of v_k;
  `// Record slicing operations of` v_k
  $v_k.\texttt{rf}_{dij} \leftarrow v_k.\texttt{take\_from}(\{u_k.\texttt{rf} \mid u_k \in \mathcal{U}_k\})$;
  `// Merge hypercubes`
  $v_k.\texttt{rf}_{dij} \leftarrow \texttt{merge}(v_k.\texttt{rf}_{dij})$;
return $\mathcal{G}$

D. Additional Pixel Space Mapping Algorithm Details

In order to compute a pixel space heat map, we propose an algorithm based on `FunctionalRF` rather than naively upsampling an embedding space similarity map $\boldsymbol{S}_{ij}$. Our approach uses the same idea as going from embedding space to pixel space. Each pixel space heat map $\boldsymbol{M}_{ij} \in \mathbb{R}^{H \times W}$ is initialized to all zeros ($\mathbf{0}^{H \times W}$), and corresponds to a sample $\boldsymbol{x}_i$ and a prototype $\boldsymbol{p}_j$. Let $\boldsymbol{M}_{ij}^S$ be the region of $\boldsymbol{M}_{ij}$ defined by the receptive field of similarity score $S \in \boldsymbol{S}_{ij}$. For each S, the pixel space heat map is updated as $\boldsymbol{M}_{ij}^S \leftarrow \max(\boldsymbol{M}_{ij}^S, S)$ where $\max(\cdot)$ is an element-wise maximum that appropriately handles the case of overlapping receptive fields. Note that we weight S by $T(|r_i|, |r_j|, \sigma_T)$ (via broadcasting) where T generates a 2D Gaussian kernel, $|\cdot|$ gives the length of a discrete interval, and σ_T is the standard deviation of the kernel. The first two arguments denote the height and width of the kernel, respectively. We set σ_T to the larger of the height and width. The intuition behind this approach is that a receptive field is actually Gaussian [19] – pixels in the center are more important, and pixels at the periphery are less important. We stress that this does not affect localization and the pixel space heat maps are largely unchanged without this

Figure B5. Examples of PIXPNET explanations on CUB-200-2011.

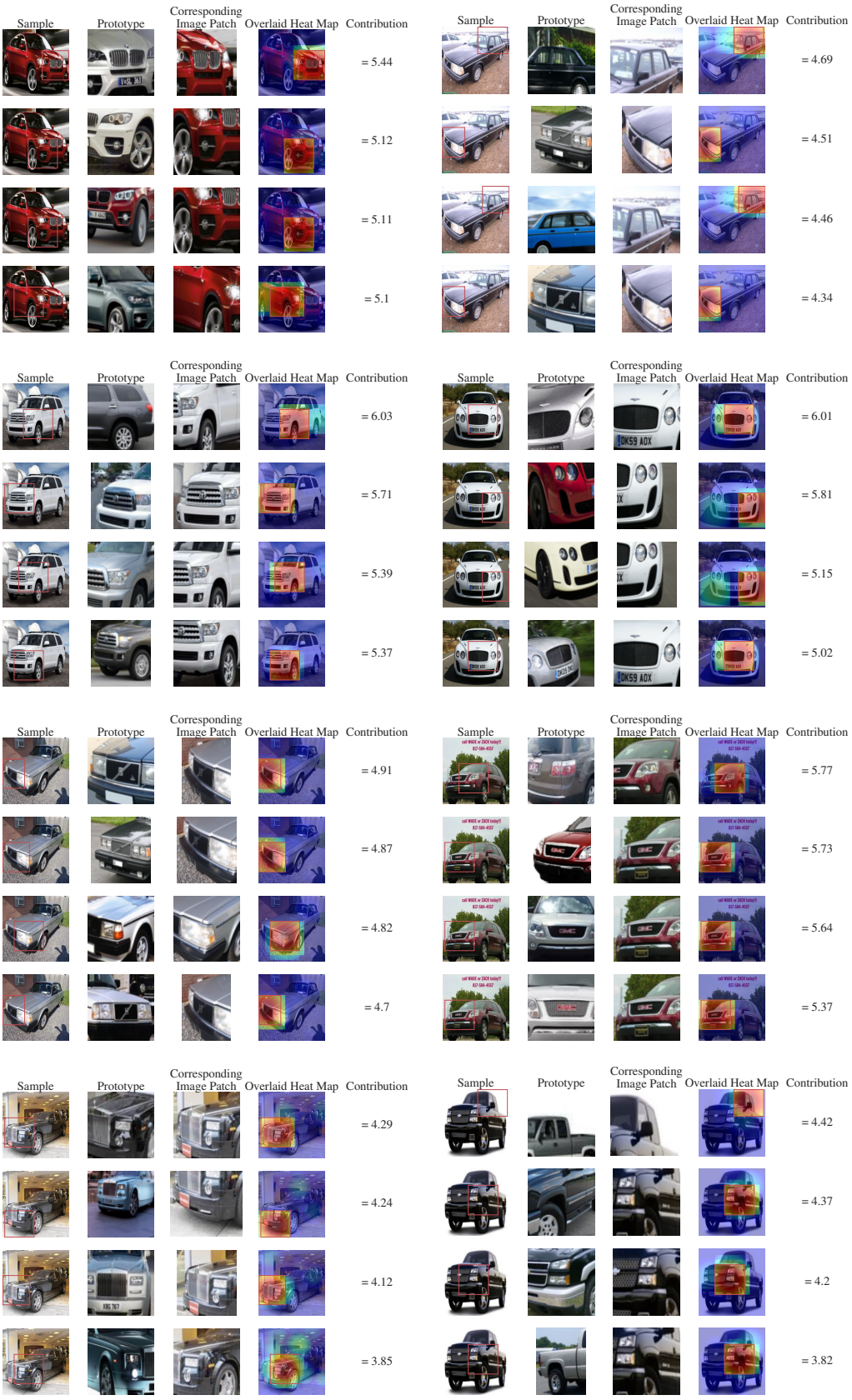

Figure B6. Examples of PIXPNET explanations on Stanford Cars.

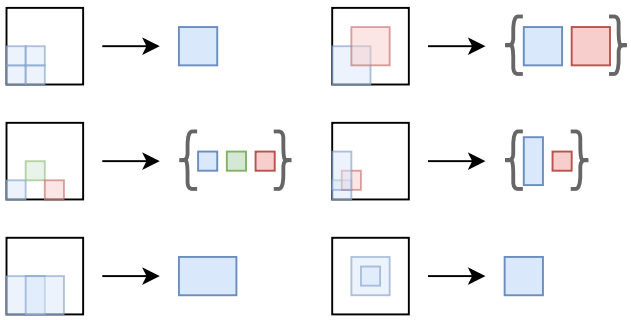

Figure C7. Examples of index merging with 2D slices. Slices that cannot be merged are retained as a set.

step. However, this does affect the relevance order testing – without this Gaussian weighting step, the pixels within a receptive field would all have the same value, so the most important pixel in the region is selected arbitrarily. Selecting in the center and "spiraling" outwards is more faithful to what is known about deep neural networks [19]. In experiments, we also confirm that this step improves the relevance ordering test scores, especially in the case of large receptive fields. The full procedure (`RFPixelSpaceMapping`) is shown in Algorithm 2.

Algorithm 2: `RFPixelSpaceMapping`(v_k, $\boldsymbol{S}$)

Input: v_k, the embedding layer in $\mathcal{G}$ (augmented by Algorithm 1)
Input: $\boldsymbol{S}$, the similarity map for a prototype
Output: $\boldsymbol{M}$, a pixel space heat map

$\boldsymbol{M} \leftarrow$ matrix of zeros $\in \mathbb{R}^{H \times W}$;
for $\texttt{rf}_{dij} \in v_k.\texttt{rf}$ **do**
  for $(r_d, r_i, r_j) \in \texttt{rf}_{dij}$ **do**
    $\boldsymbol{M}_{r_d r_i r_j} \leftarrow \max(\boldsymbol{S}_{dij} \times T(|r_i|, |r_j|, \sigma_T), \boldsymbol{M}_{r_d r_i r_j})$;
return $\boldsymbol{M}$

E. Experiment Setup and Reproducibility

Hardware and Software The code for this paper was implemented in Python and primarily relies on PyTorch [24] and PyTorch Lightning [8]. The original ProtoPNet [5] code was used to guide some development, but the majority of code deviated (especially the training code which actually uses a validation split for validation and tuning). All experiments were run on NVIDIA A10 GPUs. Unless stated elsewhere, all experimental results are reported as the average across 10 trials. Our code will be made fully available upon publication.

Data Augmentation For the CUB-200-2011 and Stanford Cars datasets, we perform the following augmentations to each training image:

1. Resize smallest dimension of image to 510 pixels with bilinear interpolation
2. Gaussian blur with 5×5 kernel and random sigma in the range $[0.1, 5]$ (1/10 probability)
3. Randomly adjust sharpness by a factor of 1.5 (1/10 probability)
4. Randomly rotate image in the range $[-15, 15]$ degrees (1/3 probability)
5. Randomly distort the image perspective with a scale of 0.2 (1/3 probability)
6. Randomly shear the image by 10 degrees (1/3 probability)
7. Randomly flip the image horizontally (1/2 probability)
8. Randomly crop the image to 384×384 pixels
9. Downsample the image to 224×224 pixels with bilinear interpolation
10. Normalize the image to have the channel-wise mean and standard deviation of the full data set

For ImageNette, steps 1 and 9 are omitted, and the random crop is done to 224×224 pixels directly.

We use an augmentation factor which indicates how many augmented samples are generated for each training image. In ProtoPNet, 30 augmentations are generated for each sample. In addition, ProtoPNet uses offline (static) augmentation, whereas we use online augmentation.

Training The training procedure largely follows that of ProtoPNet. For a warm-up period, we train just the add-on layers f_{add} and the prototypes $\boldsymbol{P}$. Thereafter, all layers are trained. We use an exponential warm-up of the learning rate and use a cosine annealing learning rate scheduler (without restarts) [18]. Every k epochs, we perform the prototype replacement procedure.

Hyperparameters All hyperparameters are listed in the proceeding table.

F. "Goldilocks" Zone Experimental Details

For this experiment, we follow the same training procedure as for PixPNet, except for any ProtoPartNN-specific training (*e.g.*, prototype replacement). See the previous section for data augmentation details. We select pre-packaged and ImageNet pre-trained architectures from PyTorch [24] to evaluate the intermediate layers of. Recall that the goal of this experiment is to discover backbones suitable for PixPNet by observing the Pareto front of the mean receptive field and accuracy on ImageNette [9]. See the main text for

Name	Value
Augmentation Factor	16
Validation Set Proportion	0.1
Pre-training	ImageNet
λ_{cls}	0
λ_{sep}	0
φ	Cosine Distance
ε	10^{-6}
P	$C \times 10$
D	192
H_p	1
W_p	1
Learning Rate (f_{core} parameters)	0.0001
Learning Rate (f_{add} parameters)	0.003
Learning Rate ($\boldsymbol{P}$)	0.003
Warm-Up Epochs	5
Learning Rate Scheduler (Warm-Up)	Exponential
Learning Rate Scheduler	Cosine
Weight Decay (All Parameters Except $\boldsymbol{P}$)	0.001
Optimizer	Adam
Batch Size	64
Epochs	20
Prototype Replacement	Every 4 Epochs

Table E4. Table of PIXPNET hyperparameters used in experiments.

discussion about this data set and justification for this approach. For each selected intermediate layer, the network is dissected at that point and a new classification head is appended which comprises a 1×1 2D adaptive average pooling layer, a flattening of the unary dimensions, and a fully-connected layer. The training procedure is carried out for 10 epochs with a batch size of 32. The proceeding table outlines the selected architectures and intermediate layers that are evaluated.

G. Similarity Function Formulation

We reduce numerical error of the original similarity function by reformulating it as:

$$v(d) = \underbrace{\log\left(\frac{d+1}{d+\varepsilon}\right)}_{\text{Original formulation}} = \underbrace{\log\left(\frac{1}{d+\varepsilon}+1\right)}_{\text{New formulation}}. \quad (1)$$

To validate its accuracy, we compare its scores across various values to the expected similarity scores with infinite precision. We use the `mpmath` [21] library to implement infinite precision. The table below compares the mean-squared-error of our approach to the original and our version of the similarity function. Our reformulation achieves lower error, especially in the $[1, 10]$ value range for both 32- and 64-bit IEEE 754 floating point numbers.

Architecture	Intermediate Layers
densenet121 densenet161 densenet169 densenet201	conv0,norm0,relu0,pool0,denseblock1, transition1,denseblock2,transition2, denseblock3,transition3,denseblock4,norm5, avgpool,classifier
inception_v3	Conv2d_1a_3x3, Conv2d_2a_3x3,Conv2d_2b_3x3,maxpool1, Conv2d_3b_1x1,Conv2d_4a_3x3,maxpool2, Mixed_5b,Mixed_5c,Mixed_5d,Mixed_6a, Mixed_6b,Mixed_6c,Mixed_6d,Mixed_6e, Mixed_7c,avgpool,fc
resnet18 resnet34 resnet50 resnet101 resnet152 resnext101_32x8d resnext101_64x4d resnext50_32x4d wide_resnet50_2 wide_resnet101_2	conv1,maxpool,layer1,layer2,layer3,layer4, avgpool,fc
squeezenet1_0 squeezenet1_1	conv1,maxpool1,maxpool2,maxpool3, features,final_conv
vgg11 vgg13 vgg16 vgg19	conv1,maxpool1,maxpool2,maxpool3, maxpool4,maxpool5,avgpool,classifier

Table F5. All evaluated intermediate layers of backbone candidate architectures.

Dtype	Region	MSE		% Improved
		Original	Ours	
Float32	0, 1e-6	1.05e-13	1.04e-13	1.11%
	1e-6, 1e-3	4.08e-14	3.97e-14	2.69%
	1e-3, 1	3.56e-15	3.30e-15	7.41%
	1, 10	1.42e-15	1.03e-15	27.13%
	10, 1000	1.19e-15	1.18e-15	0.90%
Float64	0, 1e-6	3.61e-31	3.56e-31	1.14%
	1e-6, 1e-3	1.40e-31	1.35e-31	3.09%
	1e-3, 1	1.16e-32	1.05e-32	9.12%
	1, 10	4.63e-33	3.27e-33	29.28%
	10, 1000	4.12e-33	4.09e-33	0.65%

Table G6. The comparative numerical error of the similarity functions: original and ours. Mean-squared-error (MSE) is shown for IEEE 754 floating point data types. Our reformulation is more accurate especially in the $[1, 10]$ range of values.

H. Issues with PROTOPNET Code Base

Upon inspection of the original code base[1], we discovered that the test set accuracy is used to influence training of PROTOPNET. In fact, neither PROTOPNET nor its extensions for image classification that are mentioned in the paper employ a validation set in provided implementations. Part of this is due to some code bases being derived from the original implementation. PROTOPNET peeked at test set, which

[1] https://github.com/cfchen-duke/ProtoPNet

propagated to subsequent paper implementations. According to their implementation, they did not only have no validation set, In addition, the provided code also used the accuracy on the test set to influence when training should stop.

The relevant portions of code are shown below[2]. In `main.py`, there is a training loop that saves the model twice each epoch (once before prototype replacement and once after). Each time that the model is saved, the accuracy on the test set is stored with the model. In addition, if the test accuracy is above 70%, then a message is logged to the console stating that the test accuracy is above said target.

Listing 1. `main.py` Snippet

```
accu = tnt.test(model=ppnet_multi, dataloader=
    test_loader,
                class_specific=class_specific, log=log)
save.save_model_w_condition(model=ppnet, model_dir=
    model_dir, model_name=str(epoch) + 'push', accu=
    accu,
                            target_accu=0.70, log=log)
```

The `save` module is from the `save.py`. The relevant snippet is shown below.

Listing 2. `save.py` Snippet

```
def save_model_w_condition(model, model_dir, model_name,
    accu, target_accu, log=print):
    '''
    model: this is not the multigpu model
    '''
    if accu > target_accu:
        log('\tabove {0:.2f}%'.format(target_accu * 100)
            )
        # torch.save(obj=model.state_dict(), f=os.path.
            join(model_dir, (model_name + '{0:.4f}.pth
            ').format(accu)))
        torch.save(obj=model, f=os.path.join(model_dir,
            (model_name + '{0:.4f}.pth').format(accu)))
```

The purpose of the held-out test set is to properly measure generalization error, not be used to tune hyperparameters nor influence training. Doing so is actually overfitting the distribution of the test set rather than truly improving performance. This phenomenon unfortunately has been long-standing in deep learning research, affecting progress with prominent datasets, including CIFAR-10 and ImageNet [12, 26, 27]. In our implementation, we employ a proper validation set and tune hyperparameters only according to accuracy on this split. This ensures that we are properly measuring generalization error.

In addition, we were able to approach but not reproduce the originally reported accuracies, nor could some others, using the provided code and instructions[3].

[2]These verbatim snippets are taken from the git commit `c02e8568900f20df704f65aeb86f0dd1738ca785` (most recent commit as of 2023-03-13)

[3]See `ProtoPNet` GitHub issue numbers 9, 10, 11 (the authors did not respond) and [22].

I. Full Discussion of ProtoPNet Variants and Extensions

The idea of sharing prototypes between classes has been explored in ProtoPShare [29] (prototype merge-pruning) and ProtoPool [28] (differential prototype assignment). In ProtoTree [23], the classification head is replaced by a differentiable tree, also with shared prototypes. A procedure is also proposed to convert to a hard tree to improve interpretability. An alternative embedding space is explored in TesNet [39] based on Grassmann manifolds. The authors additionally propose a new similarity function, an orthogonality loss, and a subspace loss. A ProtoPartNN-specific knowledge distillation approach is proposed in Proto2Proto [14] by enforcing that student prototypes and embeddings should be close to those of the teacher. Deformable ProtoPNet [7] extends the ProtoPNet architecture with deformable prototypes. ST-ProtoPNet [38] learns support prototypes that lie near the classification boundary and trivial prototypes that are far from the classification boundary.

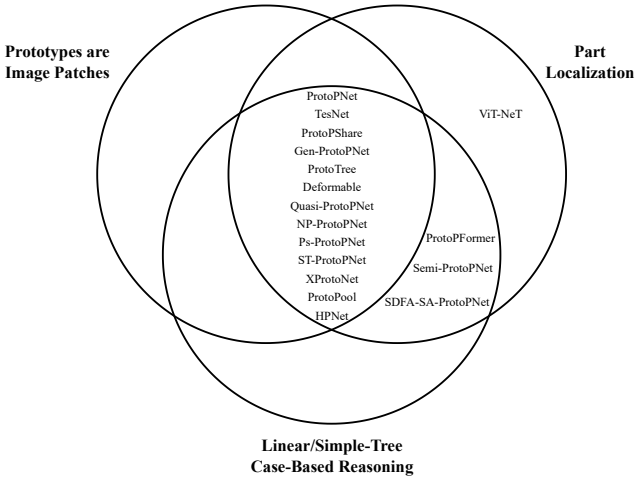

Figure I8. Proposed desiderata of ProtoPartNNs: part localization, linear/simple-tree case-based reasoning, and prototypes are image patches. Architectures at the intersection of all three desiderata are considered *true ProtoPartNNs*. Only architectures for image classification tasks are shown.

ProtoPNet has also been extended for graphs. ProtGNN [41] is an adaptation for graph classification and is built on by PxGNN [6], which extends it for node classification. Simultaneously, GCN-{TesNet,ProtoPNet} [25] was also proposed for graph and node classification. Likewise ProSeNet [20], extends the architecture for sequential data and ProtoSeg [30] can handle supervised image segmentation tasks. ProtoPNet has been adapted to task-specific applications, including EEG data (ProtoPMed-EEG [3]), Earth science data (ProtoLNet [2]), and

deepfake detection (DPNET [37]). There have been a handful of PROTOPNET extensions tailored for diagnosing the chest CT scans of COVID-19 patients This includes QUASI-PROTOPNET [31] (removes non-class weight connections in h), NP-PROTOPNET [34] (fixes weights in h to $+1$ and -1 for same- and non-class connections), GEN-PROTOPNET [32] (uses larger prototype kernel sizes), PS-PROTOPNET [33] (combines GEN-PROTOPNET and NP-PROTOPNET), and XPROTONET [15] (prototypes are compared with dynamically-sized feature patches).

ProtoPartNN Tools A few tools have also been proposed. In PROTOPDEBUG [4], a concept-level debugger for PROTOPNET is proposed in which a human supervisor, guided by model explanations, removes part-prototypes that have learned shortcuts or confounds. In [22], a methodology is developed to enhance ProtoPartNN explanations with color, hue, saturation, shape, texture, and contrast information.

In an attempt to improve PROTOPNET visualizations, an extension of layer-wise relevance propagation [1], Prototypical Relevance Propagation (PRP), is proposed to create more model-aware explanations [11]. PRP is quantitatively more effective in debugging erroneous prototypes and assigning pixel relevance than the original approach.

ProtoPartNN-Like Methods The following papers are inspired by PROTOPNET but cannot be considered to be the same class of model. This is due to not fulfilling at least one of the proposed ProtoPartNN desiderata.

VIT-NET [16] combines a vision transformer (ViT) with a neural tree decoder that learns prototypes. However, its training does not employ prototype pushing and has additional layers after the embedding layer that modify the embedding space. This breaks the mapping back to pixel space.

In another transformer-based approach, PROTOPFORMER [40] exploits the inherent architectural features (local and global branches) of ViTs. The prototype layer has both local and global prototypes, and a focal loss concentrates local prototypes on heterogeneous regions of the foreground. However, the training procedure of the method removes prototype replacement.

SEMI-PROTOPNET [36] fixes the readout weights as NP-PROTOPNET does and is applied to classification of defective structures in power distribution networks. However, the training procedure of the method also skips over prototype replacement.

In SDFA-SA-PROTOPNET [13], a shallow-deep feature alignment (SDFA) module aligns the similarity structures between deep and shallow layers. In addition, a score aggregation (SA) module aggregates similarity scores of prototypes in a class-wise manner to avoid learning inter-class information. Notably, the authors attempt to quantitatively evaluate the interpretability of prototype-based explanations rather than relying on qualitative examples as many other extensions have done. We discuss the proposed metrics in the main text. Once again, the training procedure omits prototype replacement.

Throughout each of these works, the main justification for removing prototype replacement is that it harms task accuracy.

PROTOVAE [10] is an extension of PROTOPNET for variational auto-encoders (VAEs). It outperforms PROTOPNET on a variety of classification tasks. It uses an orthogonality loss for intra-class diversity (same as TESNET). However, it does not employ prototype replacement, opting to rather use a decoder used to visualize prototypes.

Retrospective on Extensions Despite all these efforts, all extensions still have fundamental issues with object part localization, pixel space grounding, and heat map visualizations. We discuss what these issues are in detail in the main text.

J. Additional Consistency and Stability Details

For the CUB-200-2011 dataset, there are 15 object parts: back, beak, belly, breast, crown, forehead, left eye, left leg, left wing, nape, right eye, right leg, right wing, tail, and throat. However, we treat the left and right versions of an object as the same object as 1) semantically, they are the same object, 2) we do not want to penalize ProtoPartNNs for learning invariance to flips or rotations, and 3) data augmentation flips images, which makes lefts look like rights (and vice versa). In addition, owing to data preprocessing, some object parts are not visible in images that are visible in the uncropped image. Naturally, we do not penalize ProtoPartNNs for not matching with parts that are not visible.

Limitations The consistency and stability metrics ignore the localization capability of ProtoPartNNs and arbitrarily set the localization window to 72×72 pixels, centered around the localization bounding box midpoint. In addition, the threshold of 0.8 is arbitrary for a prototype to be consistent. We argue that a soft score (the average of the prototype-object coinciding frequencies without thresholding) makes far more sense. It can be interpreted as the average consistency of all prototypes, rather than the proportion of prototypes that are at least 80% consistent. When computing these soft scores, we noticed that the majority of them were in the $[0.6, 0.8]$ range. Last, the metric relies on human object part annotations – in the case of CUB-200-2011, all annotations are of bird parts. However, it is well-known that prototypes learn backgrounds and other foreground objects that are also discriminatory (such as oceans, tree branches, and human hands) [17, 35]. Prototypes may be consistent (and stable), but associated with the wrong part of the image just because of the 1) limited part annotations, and 2) the arbitrary 72×72 pixel window. Nonetheless, the metrics do allow for comparative interpretability

evaluation between ProtoPartNNs. Further information on the consistency and stability metrics is available in our code implementation as well as [13].

K. Additional Small Improvements

We noticed that the magnitude of feature maps always dictates prototype similarity when using Euclidean distance. Using cosine distance eliminates the influence of magnitude on prototype similarity.

In PROTOPNET, the incorrect gain and initialization method is used in the initialization of the convolutional weights in f_{add} before the sigmoid activation. The original implementation uses a Kaiming normal initialization with a gain that is intended for the ReLU activation. Instead, we use the Xavier normal initialization with a gain of one.

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(7):e0130140, 2015. 9

[2] Elizabeth A. Barnes, Randal J. Barnes, Zane K. Martin, and Jamin K. Rader. This looks like that there: Interpretable neural networks for image tasks when location matters. *Artificial Intelligence for the Earth Systems*, 1(3):e220001, 2022. 8

[3] Alina Jade Barnett, Zhicheng Guo, Jin Jing, Wendong Ge, Cynthia Rudin, and M. Brandon Westover. Mapping the ictal-interictal-injury continuum using interpretable machine learning. *arXiv*, 2022. 8

[4] Andrea Bontempelli, Stefano Teso, Fausto Giunchiglia, and Andrea Passerini. Concept-level debugging of part-prototype networks. In *International Conference on Learning Representations, ICLR*. OpenReview, 2023. 9

[5] Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan Su. This looks like that: Deep learning for interpretable image recognition. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Neural Information Processing Systems, NeurIPS*, pages 8928–8939, 2019. 3, 6

[6] Enyan Dai and Suhang Wang. Towards prototype-based self-explainable graph neural network. *arXiv*, 2022. 8

[7] Jon Donnelly, Alina Jade Barnett, and Chaofan Chen. Deformable ProtoPNet: An interpretable image classifier using deformable prototypes. In *Conference on Computer Vision and Pattern Recognition, CVPR*, pages 10255–10265. IEEE/CVF, 2022. 3, 8

[8] William Falcon and The PyTorch Lightning team. PyTorch Lightning, 3 2019. https://github.com/Lightning-AI/lightning. 6

[9] FastAI. Imagenette. https://github.com/fastai/imagenette, 2020. 6

[10] Srishti Gautam, Ahcene Boubekki, Stine Hansen, Suaiba Amina Salahuddin, Robert Jenssen, Marina M.-C. Höhne, and Michael Kampffmeyer. ProtoVAE: A trustworthy self-explainable prototypical variational model. In *Neural Information Processing Systems, NeurIPS*, 2022. 9

[11] Srishti Gautam, Marina M.-C. Höhne, Stine Hansen, Robert Jenssen, and Michael Kampffmeyer. This looks more like that: Enhancing self-explaining models by prototypical relevance propagation. *Pattern Recognition*, 136:1–13, 2023. 9

[12] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15262–15271, 2021. 8

[13] 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. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2011–2020, October 2023. 9, 10

[14] Monish Keswani, Sriranjani Ramakrishnan, Nishant Reddy, and Vineeth N. Balasubramanian. Proto2Proto: Can you recognize the car, the way I do? In *Conference on Computer Vision and Pattern Recognition, CVPR*, pages 10223–10233. IEEE/CVF, 2022. 3, 8

[15] Eunji Kim, Siwon Kim, Minji Seo, and Sungroh Yoon. XProtoNet: Diagnosis in chest radiography with global and local explanations. In *Conference on Computer Vision and Pattern Recognition, CVPR*, pages 15719–15728. CVF/IEEE, 2021. 9

[16] Sangwon Kim, Jae-Yeal Nam, and ByoungChul Ko. Vitnet: Interpretable vision transformers with neural tree decoder. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML*, volume 162 of *Proceedings of Machine Learning Research*, pages 11162–11172. PMLR, 2022. 3, 9

[17] Sunnie S. Y. Kim, Nicole Meister, Vikram V. Ramaswamy, Ruth Fong, and Olga Russakovsky. HIVE: evaluating the human interpretability of visual explanations. In Shai Avidan, Gabriel J. Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, *European Conference on Computer Vision, ECCV*, volume 13672 of *Lecture Notes in Computer Science*, pages 280–298. Springer, 2022. 9

[18] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. *arXiv*, 2016. 6

[19] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard S. Zemel. Understanding the effective receptive field in deep convolutional neural networks. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4898–4906, 2016. 3, 6

[20] Yao Ming, Panpan Xu, Huamin Qu, and Liu Ren. Interpretable and steerable sequence learning via prototypes. In Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis, editors, *SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD*, pages 903–913. ACM, 2019. 8

[21] mpmath Contributors. Mpmath: A python library for arbitrary-precision floating-point arithmetic, 2022. https://github.com/mpmath/mpmath. 7

[22] Meike Nauta, Annemarie Jutte, Jesper C. Provoost, and Christin Seifert. This looks like that, because... explaining prototypes for interpretable image recognition. In *Machine Learning and Principles and Practice of Knowledge Discovery in Databases - International Workshops of ECML PKDD*, volume 1524 of *Communications in Computer and Information Science*, pages 441–456. Springer, 2021. 8, 9

[23] Meike Nauta, Ron van Bree, and Christin Seifert. Neural prototype trees for interpretable fine-grained image recognition. In *Conference on Computer Vision and Pattern Recognition, CVPR*, pages 14933–14943. CVF/IEEE, 2021. 3, 8

[24] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 6

[25] Alessio Ragno, Biagio La Rosa, and Roberto Capobianco. Prototype-based interpretable graph neural networks. *IEEE Transactions on Artificial Intelligence*, pages 1–11, 2022. 8

[26] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do CIFAR-10 classifiers generalize to CIFAR-10? *arXiv*, 2018. 8

[27] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do ImageNet classifiers generalize to ImageNet? In *International conference on machine learning*, pages 5389–5400. PMLR, 2019. 8

[28] Dawid Rymarczyk, Lukasz Struski, Michal Górszczak, Koryna Lewandowska, Jacek Tabor, and Bartosz Zielinski. Interpretable image classification with differentiable prototypes assignment. In Shai Avidan, Gabriel J. Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, *European Conference on Computer Vision, ECCV*, volume 13672 of *Lecture Notes in Computer Science*, pages 351–368. Springer, 2022. 3, 8

[29] Dawid Rymarczyk, Lukasz Struski, Jacek Tabor, and Bartosz Zielinski. Protopshare: Prototypical parts sharing for similarity discovery in interpretable image classification. In Feida Zhu, Beng Chin Ooi, and Chunyan Miao, editors, *SIGKDD Conference on Knowledge Discovery and Data Mining, KDD*, pages 1420–1430. ACM, 2021. 3, 8

[30] Mikołaj Sacha, Dawid Rymarczyk, Łukasz Struski, Jacek Tabor, and Bartosz Zieliński. ProtoSeg: Interpretable semantic segmentation with prototypical parts. In *Winter Conference on Applications of Computer Vision (WACV)*, pages 1481–1492. IEEE/CVF, January 2023. 8

[31] Gurmail Singh. Think positive: An interpretable neural network for image recognition. *Neural Networks*, 151:178–189, 2022. 9

[32] Gurmail Singh and Kin Choong Yow. An interpretable deep learning model for covid-19 detection with chest x-ray images. *IEEE Access*, 9:85198–85208, 2021. 9

[33] Gurmail Singh and Kin-Choong Yow. Object or background: An interpretable deep learning model for COVID-19 detection from CT-scan images. *Diagnostics*, 11(9):1732, Sep 2021. 9

[34] Gurmail Singh and Kin Choong Yow. These do not look like those: An interpretable deep learning model for image recognition. *IEEE Access*, 9:41482–41493, 2021. 9

[35] Poulami Sinhamahapatra, Lena Heidemann, Maureen Monnet, and Karsten Roscher. Towards human-interpretable prototypes for visual assessment of image classification models. *arXiv*, 2022. 9

[36] Stéfano Frizzo Stefenon, Gurmail Singh, Kin Choong Yow, and Alessandro Cimatti. Semi-ProtoPNet deep neural network for the classification of defective power grid distribution structures. *Sensors*, 22(13):4859, 2022. 9

[37] Loc Trinh, Michael Tsang, Sirisha Rambhatla, and Yan Liu. Interpretable and trustworthy deepfake detection via dynamic prototypes. In *Winter Conference on Applications of Computer Vision, WACV*, pages 1972–1982. IEEE, 2021. 9

[38] Chong Wang, Yuyuan Liu, Yuanhong Chen, Fengbei Liu, Yu Tian, Davis McCarthy, Helen Frazer, and Gustavo Carneiro. Learning support and trivial prototypes for interpretable image classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2062–2072, October 2023. 3, 8

[39] Jiaqi Wang, Huafeng Liu, Xinyue Wang, and Liping Jing. Interpretable image recognition by constructing transparent embedding space. In *International Conference on Computer Vision, ICCV*, pages 875–884. IEEE/CVF, 2021. 3, 8

[40] Mengqi Xue, Qihan Huang, Haofei Zhang, Lechao Cheng, Jie Song, Minghui Wu, and Mingli Song. ProtoPFormer: Concentrating on prototypical parts in vision transformers for interpretable image recognition. *arXiv*, 2022. 3, 9

[41] Zaixi Zhang, Qi Liu, Hao Wang, Chengqiang Lu, and Cheekong Lee. ProtGNN: Towards self-explaining graph neural networks. In *Conference on Innovative Applications of Artificial Intelligence, IAAI*, pages 9127–9135. AAAI Press, 2022. 8