Supplementary Material for SPARROW: Semantically Coherent Prototypes for Image Classification

Stefan Kraft[1,4]
stefan.kraft@stz-softwaretechnik.de
Klaus Broelemann[2]
klaus.broelemann@schufa.de
Andreas Theissler[3]
https://orcid.org/0000-0003-0746-0424
Gjergji Kasneci[4,2]
gjergji.kasneci@uni-tuebingen.de

[1] IT-Designers Group
Esslingen am Neckar, GER
[2] SCHUFA Holding AG
Wiesbaden, GER
[3] Aalen University of Applied Sciences
Aalen, GER
[4] Data Science & Analytics Research
The University of Tübingen
Tübingen, GER

This supplementary material is structured as follows. We start by summarizing the mathematical notation in section 1 and proceed to give additional details about the model architecture (section 2), training process (section 3), `SPARROW` framework (section 4) and experiments (section 5) presented in the main paper.

1 Notation

Table 1 summarizes the conventions regarding the notation and meaning of mathematical symbols in the main paper and the remainder of this supplementary material.

2 Model architecture

This section provides additional details about the model architecture of ProtoPNet [1]. An overview is shown in table 2. The prototype classification network learns a transformation $h \circ g_P \circ f(X)$ on images from a given dataset $D = [X,Y] = \{\mathbf{x}_i, y_i\}_{i=1}^n$. The transformation is composed of a convolutional base network f, a prototype unit g_P and a fully-connected layer h.

Convolutional base network The convolutional base network f learns a latent space feature embedding. We select a ResNet18 architecture to which we append two convolutional addon-layers which do not change the shape of the ResNet18 output in accordance with Chen *et al.* [1]. For the ResNet18 layers we use weights pretrained on ImageNet [5]. f reduces the sample dimensions from S_{raw} to $S = (s,s) = (7,7)$ and the whole latent space dimension is $(l,s,s) = (512,7,7)$, where l is the number of convolutional filter layers.

Symbol	Meaning
n_{train}	Number of unaugmented training samples
n_{test}	Number of test samples
N	Total number of unaugmented samples in the dataset. $N = n_{\text{train}} + n_{\text{test}}$
n_{aug}	Number of augmented samples
$\rho_{\text{train-val}}$	Fraction of augmented samples n_{aug} which are used for training
n	Number of augmented training samples, given by $n = \rho_{\text{train-val}} \cdot n_{\text{aug}}$
n_{val}	Number of augmented validation samples, given by $n_{\text{val}} = (1 - \rho_{\text{train-val}}) \cdot n_{\text{aug}}$
S_{raw}	Size of samples after preprocessing (cropping and rescaling)
S	Kernel size of samples in latent space which is quadratic: $S := (s, s)$
S_{pt}	Kernel size of prototypes in latent space which is quadratic: $S_{\text{pt}} := (s_{\text{pt}}, s_{\text{pt}})$
l	Number of convolutional filter layers in latent space
m	Number of prototypes before pruning
C	Number of classes in the dataset
$\mathbf{x}_i$	Single preprocessed training sample, $i \in [1, \dots, n]$
c_i	Class of sample $\mathbf{x}_i$
X	Set of all preprocessed training samples$\{\mathbf{x}_{i=1}^{n}\}$
$\mathbf{p}_j$	Single prototype, $j \in [1, \dots, m]$
P	Set of all prototypes $\{\mathbf{p}_{j=1}^{m}\}$
P_{c_i}	Set of prototypes $\{\mathbf{p}_{j=\tilde{m}_i}^{\tilde{m}_i\prime}\}$ of class c_i with $\tilde{m}_i, \tilde{m}_i\prime \in [1, \dots m]$
$\mathcal{P}_{c_i}$	Interval of indices $[\tilde{m}_i, \dots, \tilde{m}_i\prime]$ for prototypes of class c_i with $\tilde{m}_i, \tilde{m}_i\prime \in [1, \dots m]$
f	Combination of base- and add-on layers of the neural network
g_P	Prototype unit of the neural network which contains prototypes P
h	Last layer of the neural network
$\mathcal{L}$	Total loss in the prototype learning phase of the training
$\mathcal{L}^{\nu}$	ν-th loss component of $\mathcal{L}$ with $\nu \in \{\text{CrsEnt}, \text{Clst}, \text{Sep}, \text{AS}, \text{PSD}\}$
$\lambda_{\mathcal{L}^{\nu}}$	Weight of the ν-th loss component $\mathcal{L}^{\nu}$ in the total loss $\mathcal{L}$
$\omega_h^{\gamma,j}$	Last layer weights with $\gamma \in [1, \dots, C]$ and $j \in [1, \dots m]$
$\mathcal{L}^{L1}$	$L1$ norm regularization of last layer weights $\omega_h^{\gamma,j}$
$\lambda_{\mathcal{L}^{L1}}$	Weight of the regularizer $\mathcal{L}^{L1}$ in the total last layer optimization loss
T	Number of annotated part types in the dataset
$\tilde{T}_i$	Number of parts in the i-th sample which are matched by at least one prototype
$\mathbf{r}_{i,k}$	Coordinate of the k-th part in the transformed sample input space of the i-th sample
R	Set of part coordinates $\{\mathbf{r}_{i,k=1,1}^{N,T}\}$
$\text{match}_{i,j,k}$	Scalar which is 1 if there is a match between the j-th prototype ($j \in \mathcal{P}_{c_i}$) and the k-th part ($k \in [1, \dots, T]$) for the i-th sample ($i \in [1, \dots, N]$). Otherwise it is 0

Table 1: **Table of notation**

Hierarchy	Type	Output size	Info
Base	ResNet	$(512,7,7)$	ResNet architecture [8], based on the torchvision implementation [9], where we stripped away the last two layers (average pooling and fully connected layer)
Add-on	conv	$(512,7,7)$	Kernel size $(1,1)$, stride $(1,1)$, ReLU activation function, Kaiming He initialization [2]
Add-on	conv	$(512,7,7)$	Kernel size $(1,1)$, stride $(1,1)$, Sigmoid activation function, Kaiming He initialization [2]
Prototype	params	$(512,s_{\text{pt}},s_{\text{pt}})$	Kernel size $(s_{\text{pt}},s_{\text{pt}})$, parameters are randomly initialized from a uniform distribution on $[0,1)$
Last-layer	fc	(m,C)	No bias, Kaiming He initialization [2]

Table 2: **Model architecture** of ProtoPNet

Prototype unit The prototype unit g_P contains a set of m prototypes $P = \{\mathbf{p}_{j=1}^{m}\}$ which occupy a latent space kernel of the minimal possible size $S_{pt} = (s_{pt}, s_{pt}) = (1,1)$ and therefore have the shape (m,l,s_{pt},s_{pt}). For each sample, g_P first calculates the squared $L2$ distances between the latent space patches of samples $f(X)$ and the prototypes P. This calculation is performed per prototype while summing over the filter layers. The resulting distances are converted to similarity scores. Concretely, this means that the j-th prototype unit $g_{\mathbf{p}_j}$ computes a similarity score between prototype $\mathbf{p}_j$ and latent space patches of a sample $\xi \in X$ as [1]

$$g_{\mathbf{p}_j}(f(\xi)) = \max_{\mathbf{z} \in patches(f(\xi))} \log\left(\frac{\|\mathbf{z}-\mathbf{p}_j\|^2+1}{\|\mathbf{z}-\mathbf{p}_j\|^2+\varepsilon}\right), \tag{1}$$

where ε is a very small number. Since this is a monotonically decreasing function with respect to the $L2$ distance $\|\mathbf{z}-\mathbf{p}_j\|$ between a prototype and a latent space patch of a sample, it is a suitable choice for describing the similarity between a prototype and sample patch [1]. This means that the similarity between a sample and a prototype is effectively determined by the latent space patch of the sample which is most similar to the given prototype. The similarity scores have the shape (m,s_{pt},s_{pt}) which in our case is $(m,1,1)$.

Fully connected layer The fully connected layer h (called "last layer") calculates final class scores as a weighted superposition of the output of g_P. This is done by applying a fully connected layer without a bias term on the resulting similarity scores which is of shape (m,C) where C is the number of classes.

3 Training process

In this section we provide further details about the training process of ProtoPNet [1].

Training steps The training consists of three main steps. As a final step prototypes that focus on background semantics may be pruned, *e.g.* with the approach by Chen *et al.* [1]. In this work we do not perform any pruning.

Step 1: Prototype learning Parameters of the neural network and the prototypes are jointly learned while the weights of the last layer are frozen. An important concept is that the weights of the last layer are fixed such that each output logit is connected to an equal

number of outputs of g_P while all other connections are set to -0.5. For this reason it can be said that prototypes "have a certain class identity" or "belong to a class". The total loss function is composed of a weighted superposition of five loss components with static weights as (cf. table 1 in the main paper): $\mathcal{L} = \sum_\nu \lambda_{\mathcal{L}^\nu} \cdot \mathcal{L}^\nu$ with $\nu \in \{\text{CrsEnt}, \text{Clst}, \text{Sep}, \text{AS}, \text{PSD}\}$. Only class labels but no part annotations are used in training.

Step 2: Prototype projection After the main training is finished a prototype projection (also called "prototype push") is performed by which the prototypes are moved in latent space to the sample patch with the same class identity as the prototype which has the minimum $L2$-distance of all samples in the training set.

Step 3: Last layer optimization The prototype push is followed by a convex optimization of the last layer where the weights of all other layers are frozen. During this phase a $L1$ norm regularization of the off-label weight connections $\omega_h^{\gamma,j}$ between the C classes ($\gamma \in [1,\dots,C]$) and m outputs ($j \in [1,\dots m]$) of the prototype unit g_P is performed as

$$\mathcal{L}^{L1} = \sum_\gamma \sum_{j \notin \mathcal{P}_\gamma} |\omega_h^{\gamma,j}|, \tag{2}$$

where $\mathcal{P}_\gamma$ is the set of prototypes of the γ-th class. The off-label weights which were initially fixed to -0.5 are effectively reduced to zero. This is done to prevent a negative reasoning process for the classification [1]. This loss is added to the total loss (which at this stage is given by $\mathcal{L}^{\text{CrsEnt}}$) with weight $\lambda_{\mathcal{L}^{L1}}$. The last layer optimization is performed over $\text{epochs}_{\text{ll}}$ epochs.

Train-validation split Before the training, we perform a train-validation split where we split the n_{aug} augmented samples into $n = (\rho_{\text{train-val}}) \cdot n_{\text{aug}}$ training and $n_{\text{val}} = (1 - \rho_{\text{train-val}}) \cdot n_{\text{aug}}$ validation samples, where $\rho_{\text{train-val}}$ is the splitting factor.

Warm start The training starts with a "warm start" of several epochs in which only the parameters of the add-on layers and the prototype vectors P are optimized while the parameters of the base network and the last layer are frozen. Only prototype learning is performed during the warm start. The training phase after the warm start is refered to as "main training".

Early stopping Both the warm start and main training run until stopped by an early-stopping method when the average validation accuracy does not increase by the fraction $\text{delta}_{\text{warm/main}}$ of the validation score over $\text{pat}_{\text{warm/main}}$ epochs (patience). After early stopping of the prototype learning step in the main training phase, the model checkpoint of pat_{main} epochs earlier is loaded and used for the following steps, i.e. prototype push and last layer optimization.

Clamping of cosine similarity The cosine similarity $\text{CS}(\mathbf{p}_i, \mathbf{p}_j) = \frac{\mathbf{p}_i^T \cdot \mathbf{p}_j}{\|\mathbf{p}_i\| \|\mathbf{p}_j\|}$ between two prototypes $\mathbf{p}_i$ and $\mathbf{p}_j$ can theoretically take its maximum or minimum value of 1 and -1. Especially the former will occasionally happen after a prototype push if two prototypes are pushed to the same latent space sample patch of the same sample. This is not a problem in principal [1] but will lead to infinite gradients of $\text{AS}(\mathbf{p}_i, \mathbf{p}_j) = 1 - \frac{1}{\pi} \arccos(\text{CS}(\mathbf{p}_i, \mathbf{p}_j))$ during backpropagation. To prevent this we decided to clamp the values of the cosine similarity to the interval $[-0.9999, 0.9999]$.

Simplifications In order to simplify the training process and since we are not interested in optimizing the classification performance of the baseline in the context of this work we only perform one final push epoch and also do not use a learning rate scheduler in contrast to Chen *et al*. [1].

4 SPARROW

Matching Prototypes with Part Annotations This section gives a more detailed explanation on how to obtain the matches between prototypes and part annotations (cf. section 2 of the main paper). We define that the part annotations $R = \{\mathbf{r}_{i,k=1,1}^{N,T}\}$ have already been subject to the same transformations as the raw input samples (cf. section 5). As noted in algorithm 1 we first obtain the activation map of the latent space sample patch which has the smallest $L2$ distance to a prototype (line 5). This is done for all samples $\{\mathbf{x}_{i=1}^{N}\}$ and prototypes $P_{c_i} = \{\mathbf{p}_{j=\tilde{m}_i}^{\tilde{m}_i\prime}\}$ with the same class identity $c_i \in [1,\dots,C]$ as the samples. Here, $\tilde{m}_i, \tilde{m}_i\prime \in [1,\dots m]$ and we define $\mathcal{P}_{c_i} = [\tilde{m}_i,\dots,\tilde{m}_i\prime]$ to be the interval of indices of prototypes which belong to class c_i. The activation maps are upsampled to the pixel space of sample inputs (line 6). Next, a mask is selected based on an activation threshold (line 7).

Schematic prototype masks are illustrated together with annotated image parts in figure 1. Each part coordinate is matched with the mask and matches are noted in $\{match_{i,j,k=1,\tilde{m}_1,1}^{N,\tilde{m}_N,T}\}$ (lines 8-13). If an activation mask does not match any part annotation, we select the closest coordinate to the mask as a match (lines 14-17).

Algorithm 1 Find matches between sample patch activations by prototypes and part annotations.

1: $\text{match}_{i,j,k} \leftarrow$ initialize to 0 for $(i,j,k) = (1,\tilde{m}_1,1),\dots,(N,\tilde{m}_N,T)$
2: **for** $i = 1,2,\dots,N$ **do**
3: **for** $j = \tilde{m}_i\dots,\tilde{m}_i\prime$ **do**
4: pt_has_match$\leftarrow$ *False*
5: $\text{act}_{i,j} \leftarrow \min_{\mathbf{z}\in patches(f(\mathbf{x}_i))} \|\mathbf{z}-\mathbf{p}_j\|^2$
6: $\text{act}_{i,j}^{up} \leftarrow$ upsampling of $\text{act}_{i,j}$ to pixel space
7: $\text{mask}_{i,j} \leftarrow$ select values $>$ threshold in $\text{act}_{i,j}^{up}$
8: **for** $k = 1,2,\dots,T$ **do**
9: **if** $\mathbf{r}_{i,k}$ lies within the mask **then**
10: $\text{match}_{i,j,k} \leftarrow 1$
11: pt_has_match$\leftarrow$ *True*
12: **end if**
13: **end for**
14: **if** not pt_has_match **then**
15: Find k for which $\mathbf{r}_{i,k}$ is closest to the mask
16: $\text{match}_{i,j,k} \leftarrow 1$
17: **end if**
18: **end for**
19: **end for**

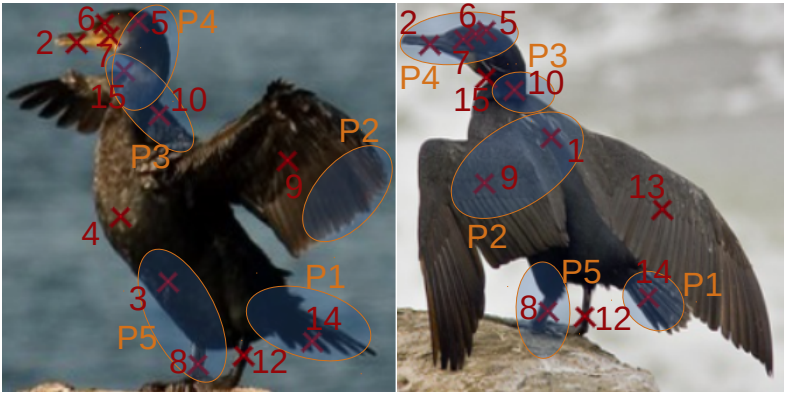

Figure 1: **Part annotations vs. activation masks.** Sample images from the CUB [6] dataset (containing $T = 15$ part annotations) showing annotations of visible parts (numbers 1 to 15) and schematic illustrations of the activation masks of prototypes (P1 to P5).

5 Experiments

This section provides additional details about the experiments that we performed (cf. section 3 of the main paper).

Dataset The Caltech-UCSD Birds-200-2011 dataset (CUB) [6] contains $N = 11788$ images of birds which divide into $n_{\text{train}} = 5994$ training and $n_{\text{test}} = 5794$ test samples. For each sample there exists one class- and $T = 15$ keypoint part annotations[1], where the total number of classes is $C = 200$ bird species. The samples are distributed reasonably evenly between classes so that one class contains about 30 samples. After preprocessing which includes data augmentation we have $n = 194206$ augmented training samples of size $S_{\text{raw}} = (224, 224)$ and each class contains about 1200 images. Training is performed on the augmented training data while the protoype projection uses unaugmented training samples and the `SPARROW` evaluation protocol is applied on all unaugmented training and test samples.

Data Preprocessing Preprocessing of the samples from the CUB dataset follows the approach from Chen *et al*. [1] and consists of the following steps in the given order:

- The birds are cropped from the background by bounding boxes which are provided by the dataset.
- All samples are rescaled to size $S_{\text{raw}} = (224, 224)$.
- Data augmentation is applied on all training samples which consists of rotating, skewing, shearing, randomly distorting and flipping the images.
- All samples are normalized by the mean and standard deviation of the training set.

[1] Parts are: back, beak, belly, breast, crown, forehead, left eye, left leg, left wing, nape, right eye, right leg, right wing, tail, throat. Not all parts are visible in every sample.

Context	Parameter	Value	Explanation
Data (CUB)	n_{train}	5994	
	n_{test}	5794	
	N	11788	
	n_{aug}	215784	
	$\rho_{\text{train-val}}$	0.9	
	n	194206	
	n_{val}	21578	
	C	200	
	T	15	
	S_{raw}	$(224, 224)$	
Model	S_{pt}	$(1, 1)$	
	m	$\{1000, 2000, 4000\}$	Number of prototypes used in the experiments
Training	$\lambda_{\mathcal{L}^{\text{CrsEnt}}}$	1	
	$\lambda_{\mathcal{L}^{\text{Clst}}}$	1	
	$\lambda_{\mathcal{L}^{\text{Sep}}}$	0	
	$\lambda_{\mathcal{L}^{\text{AS}}}$	0 / 1	for baseline / own method
	$\lambda_{\mathcal{L}^{\text{PSD}}}$	0 / 100	for baseline / own method
	$\lambda_{\mathcal{L}^{L1}}$	1e−2	
	batch size	32	Number of samples per mini-batch in a training iteration
	optimizer	Adam [4]	
	lr_{base}	1e−4	Learning rate in the base network (cf. table 2)
	$\text{lr}_{\text{add-on}}$	1e−4	Learning rate in the add-on network (cf. table 2)
	$\text{lr}_{\text{prototype}}$	3e−3	Learning rate in the prototype layer (cf. table 2)
	$\text{decay}_{\text{base}}$	1e−3	Weight decay of the Adam opt. in the base network
	$\text{decay}_{\text{add-on}}$	1e−3	Weight decay of the Adam otp. in the add-on network
	pat_{warm}	1	Warm start early stopping patience in epochs
	pat_{main}	3	Main training early stopping patience in epochs
	$\text{delta}_{\text{warm}}$	1e−2	Warm start early stopping criterion
	$\text{delta}_{\text{main}}$	5e−3	Main training early stopping criterion
	$\text{epochs}_{\text{ll}}$	3	Epochs of convex optimization of the last layer

Table 3: **Choices for Hyperparameters**. The meaning of the symbols is explained in table 1.

With $\mathcal{L}^{\text{AS}}$	With $\mathcal{L}^{\text{PSD}}$	$\text{epochs}_{\text{warm}}$	$\text{epochs}_{\text{main}}$	time [h]	acc_{val} [%]
False	False	10	6	6.88	98.6
False	True	9	9	8.00	99.0
True	False	10	19*	72.23*	90.7*
True	True	10	8	34.42	97.2

Table 4: **Convergence and training duration** in epochs and hours with the hardware as described in section 5 and hyperparameters in table 3. The time calculation includes all training steps (prototype learning, prototype push, last layer optimization), phases (warm start and main training) and the performance evaluation. acc_{val} is the final validation accuracy. Entries marked with * are not final since the respective experiment did not finish until the submission of this supplementary material.

Hyperparameter Optimization An overview over all hyperparemeters is shown in table 3. Most parameters are oriented on the work of Chen *et al.* [1]. In order to tune the static weights of the loss components $\lambda_{\mathcal{L}^\nu}$ ($\nu \in \{\text{CrsEnt}, \text{Clst}, \text{Sep}, \text{AS}, \text{PSD}\}$), we performed two rounds of subsequent random search and Bayesian Optimization (BO) runs. For BO we used an Upper Confidence Bound acquisition function and a Gaussian Process with a Matern kernel as a surrogate. We optimized for a measure of balance between performance and interpretability for which we used

$$\text{opt} = \frac{2 \cdot \text{acc}_{\text{val}} + \text{ptd} + \text{ptf}}{4}, \tag{3}$$

with the validation accuracy acc_{val} and the *ptd* and *ptf* measures as defined in section 2 in the main paper. We reduced the time demand of the tuning process by randomly reducing the number of classes to 20 in each experiment and ran each experiment until early stopping.

In both optimization rounds, we ran 30 initial random searches which seeded 45 iterations of BO. In the first round, we selected values for each loss component weight from an even distribution of 100 values on a logarithmic scale over the interval $[0.01, 100]$. We found that for the top performing runs, $\lambda_{\mathcal{L}^{\text{AS}}}$ was typically smaller than the weight of the other loss components except for $\lambda_{\mathcal{L}^{\text{Sep}}}$ which was typically optimized to $\lambda_{\mathcal{L}^{\text{Sep}}} = 0.01$ with all other loss component weights being at least two orders of magnitude larger. We performed a second optimization round for fine-tuning in which we set $\lambda_{\mathcal{L}^{\text{Sep}}} = 0.01$ and $\lambda_{\mathcal{L}^{\text{AS}}} = 1$ and selected the other loss component weights from an even distribution of 15 values on a logarithmic scale over the interval $[1, 100]$. Amongst the top performing runs we opted for a simple choice for the set of remaining parameters ($\lambda_{\mathcal{L}^{\text{CrsEnt}}} = 1$, $\lambda_{\mathcal{L}^{\text{Clst}}} = 1$, $\lambda_{\mathcal{L}^{\text{PSD}}} = 100$). We did a final round of experiments where we investigated if we could do without one of the loss components and actually found that setting $\lambda_{\mathcal{L}^{\text{Sep}}} = 0$ slightly improved our optimization goal (cf. eq. 3). This might be because the newly introduced *PSD* loss which tries to keep prototypes close to samples of the same class fulfills a similar purpose (i.e. keeping samples away from prototypes of other classes).

Training hardware Experiments were performed on single NVIDIA GeForce RTX 1080 Ti GPUs with 11 GB GDDR6 of graphics memory.

Convergence and training duration We analyzed how the training duration varies based on inclusion or exclusion of the new loss components $\mathcal{L}^{\text{AS}}$ and $\mathcal{L}^{\text{PSD}}$. The results are reported

Prototypes	Total runtime [h]
1000	5.25
2000	10.57
4000	19.35

Table 5: **Time demand for the SPARROW evaluation protocol** for different numbers of prototypes with the hardware as described in section 5 and hyperparameters in table 3.

in table 4. The ProtoPNet model without any additional loss components (first line) has the shortest runtime until convergence. Including the $\mathcal{L}^{\text{PSD}}$ loss component (second line) only slightly increases time to convergence. However, including the $\mathcal{L}^{\text{AS}}$ loss component (third line) but not the $\mathcal{L}^{\text{PSD}}$ loss component leads to a very slow convergence and as a result to a greatly increased time demand until early stopping. This experiment did not finish until the submission of this supplementary material but the slow convergence is clearly visible. This slow convergence is mitigated to a reasonable extent by also including the $\mathcal{L}^{\text{PSD}}$ loss component (last line) which is our proposed method. Its runtime is about five times as high as that of the original ProtoPNet model (first line).

Optimizing the time-demand of our method and developing new methods with reduced time-demand but comparable explanatory capacity (measured by the SPARROW evaluation protocol) are goals we want to pursue in future work.

Time demand for the SPARROW evaluation protocol The time-demand of the SPARROW evaluation protocol for different numbers of prototypes is shown in table 5. It can be seen that it increases about linearly with the number of prototypes. These numbers should however be seen as an easy to reach upper limit since we did not yet optimize our implementation for performance. We see this as a goal for future work.

References

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

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

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

[4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style,

high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS) 32*. 2019.

[6] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.