IMPERIAL

Acceleration of Interpretable Deep Neural Networks

Author

Yanjinlkham Temuujin

CID: 02519243

Supervised by

Prof. Wayne Luk

Dr Ce Guo

Independent Study Option Report

Department of Computing

Imperial College London

2025

Abstract

Interpretable deep learning models are essential for critical domains such as medical imaging, where transparency and trust in decision-making are crucial. Unlike conventional deep neural networks, which function as ”black boxes”, interpretable models provide insights into their reasoning process, enhancing their reliability. One such model, the Prototypical Part Network (ProtoPNet), has gained significant attention for its ability to improve interpretability in computer vision tasks. Numerous extensions have been developed based on its foundational principles to further enhance both interpretability and performance.

However, the practical deployment of these prototypical-part networks is limited by their computational inefficiency during training loss computation. This computational burden restricts their usability, making them impractical for real-time adaptation, training, and fine-tuning in various domain-specific applications.

This work presents an approach to accelerate training loss computation, significantly reducing training time by a factor of 1.05× to 3.2× while preserving model accuracy. The proposed acceleration is applicable to a broad range of prototype-based interpretable models relying on the same loss calculation, improving their practicality for real-world applications such as medical imaging and other critical domains. The implementation of the acceleration and experiment results are available at `https://gitlab.doc.ic.ac.uk/yt2623/fasterprotopnet`.

Contents

1

Introduction

1.1 Interpretability approaches

Deep neural networks have been widely utilized in computer vision tasks since the advent of convolutional neural networks, beginning with AlexNet and the large-scale ImageNet dataset. However, despite their remarkable performance, most deep neural networks operate as black-box models, offering little to no insight into their internal decision-making processes. This lack of interpretability makes them difficult to deploy in high-stakes domains such as medical imaging, where model transparency and trustworthiness are critical. To address this issue, significant research has focused on developing interpretable deep learning models. These efforts can broadly be categorized into *post-hoc* and *ante-hoc* approaches.

1.1.1 Post-hoc interpretability

Post-hoc methods seek to explain the reasoning of an already trained black-box model. Various techniques have been proposed in this category, including activation maximization, which generates synthetic images that maximize the activation of specific neurons (1, 2, 3), deconvolution methods (4), and saliency visualization techniques that use gradients to highlight the most important input features for a model's decision (5, 6, 7). Concept activation methods (8) leverage auxiliary datasets to determine which high-level concepts contribute to classification decisions. Although these post-hoc techniques provide valuable insights into model behaviour, they do not faithfully represent the internal decision-making process of the network (9), as they are derived after training rather than being inherently embedded in the model's architecture.

1.1.2 Ante-hoc interpretability

Ante-hoc methods, in contrast, are inherently interpretable by design. Among the most prominent approaches in this category are case-based deep learning models, which provide explanations based on learned prototypes. This research direction began with Prototypical Part Networks (ProtoPNet) (9) and has since given rise to numerous extensions (10, 11, 12, 13, 14, 15, 16, 17, 18, 19). These models have been applied to a wide range of domains, including breast cancer detection (11), Alzheimer's disease diagnosis (20), and deepfake detection (21). Several of these studies (12, 19, 10, 17), have demonstrated that case-based deep learning models achieve accuracy comparable to, or even surpass, that of black-box models while maintaining interpretability – an essential feature lacking in traditional deep networks (22). Moreover, prototype-based learning is highly adaptable and can be integrated with various backbone architectures, from convolution-based networks to vision transformers (17, 18)

1.2 Challenges and motivation

Despite their advantages, prototype-based models suffer from a major limitation: computational inefficiency during training. Regardless of the underlying backbone architecture, the **training process remains slow due to the high computational cost associated with similarity and loss function calculations**, which can account from 10% to 50% of the total training time per batch iteration. This inefficiency severely limits the scalability and practicality of these models, making them unsuitable for adaptation, fine-tuning for down-stream tasks, or deployment in large-scale applications.

This work aims to **accelerate the training of prototype-based models by simplifying the computation of training loss**. The main contributions of this study can be summarized as follows:

- Comprehensive analysis of existing prototype-based models architecture and their training algorithm.
- A novel acceleration strategy that moderately reduces training time while maintaining model accuracy.
- Empirical evaluation of the proposed acceleration approach to assess its effectiveness, scalability and interpretability.

2

Background and Related Work

2.1 Prototype-based model architecture

Table 2.1: Comparison of different prototype-based model architectures.

Layer	**IAIA-BL**(11)	**ProtoPNet**(9)	**TesNet**(19)	**Def.-P.**(10)	**ProtoViT**(17)
Encoder	VGG-16	VGG-16,19 ResNet-34,152 DenseNet-121,161			ViT DeiT-Tiny,Small CaiT-XXS 24
Prototype	non-deformable			deformable	
	$sim(d_{L^2})$		$sim_{\cos}$		
	top-k avg. pool	global max pool			greedy summing
Classifier	2 FC layer	1 FC layer			

While numerous extensions and variations of ProtoPNet have been proposed, they generally adhere to a common architectural framework. Although the specific implementation details may vary, the fundamental components of these models remain consistent. Formally, let $X \in \mathbb{R}^{C\times H\times W}$ represent an input image of height H, width W, and C channels, with Y denoting its class label. A prototype-based model consists of the following three key components (22).

2.1.1 Encoder

The first stage of the model applies an embedding function f that extracts a D-dimensional feature representation from the input image X. This results in a spatial feature map of dimensions $H'\times W'$, capturing high-level patterns essential for classification. Various deep learning vision backbones can be applied here, from convolution based networks, such as ResNet, VGG, DenseNet, to transformer-based models such as ViT, DeiT, CaiT.

$$f : \mathbb{R}^{C\times H\times W} \rightarrow \mathbb{R}^{D\times H'\times W'}$$

2.1.2 Prototype layer

In the prototype layer g, a model learns M prototypes, represented as $\mathbf{P} = \{p_j\}_{j=1}^{M}$. A fixed number of M_k prototypes is assigned to each class $k \in \{1, ..., K\}$ before training (9). The subset of prototypes assigned to each class k, denoted as $\mathbf{P}_k = \{p^{(k,j)}\}_{j=1}^{M_k} \subseteq \mathbf{P}$, is responsible for capturing the most distinctive features needed to correctly classify images of that class.

$$g : \mathbb{R}^{D \times H' \times W'} \to \mathbb{R}^{M \times H'' \times W''}$$

ProtoPNet and other prototype-based models (e.g. IAIA-BL (11), TesNet (19), ProtoConcepts (13)) generally use rigid or **non-deformable prototypes** $p^{(k,j)}$, each having dimensions $H'' \times W'' \times D$, where $H'' \leq H'$ and $W'' \leq W'$. In recent works, such as Deformable-ProtoPNet (10) and ProtoViT (17), spatially flexible and **deformable prototypes** are used, represented as $p_{m,n}^{(k,j)}$, denoting the (m, n)-th prototypical part within the deformable prototype $p^{(k,j)}$ of shape $H'' \times W'' \times D$. Here, each spatial position (m, n) is viewed as an "individual prototypical part that can move around, and represents a semantic concept that is spatially decoupled from other prototypical parts" (10). Further, **learned offsets** $\Delta h, \Delta w$ are introduced to compare each prototypical part $p_{m,n}^{(k,j)}$ to an image feature at a deformed position $(h + m + \Delta h, w + n + \Delta w)$, rather than at the standard rigid position $(h + m, w + n)$.

Given an output from the encoder layer, $z = f(X)$, similarity is computed between each patch $\tilde{z} \in \text{patches}(z)$ and prototype $p^{(k,j)}$. The main similarity functions that are used in prototype-based models are the following:

- L^2 **distance (Euclidean distance)** is calculated as the square root of the sum of the squared differences between corresponding elements of the given feature and prototype vectors:

$$d_{L^2}(\tilde{z}, p) = -\|\tilde{z} - p\|_2^2 = \sum_{d=0}^{D} (\tilde{z}^{(d)} - p^{(d)})^2,$$

  where the summation runs over all feature dimensions D. L^2 distances are often transformed into similarity scores using: $sim(\tilde{z}, p) = \log \frac{d_{L^2}(\tilde{z},p)+1}{d_{L^2}(\tilde{z},p)+\epsilon}$ (9, 11).

- **Cosine similarity** measures the angle between the feature vector and the prototype, and often used in models with deformable prototypes like Deformable-ProtoPNet (10) and ProtoViT (17):

$$sim_{\cos}(\tilde{z}, p_{m,n}^{(k,j)}) = \frac{\tilde{z}^T p_{m,n}^{(k,j)}}{\|\tilde{z}\|_2 \|p_{m,n}^{(k,j)}\|_2}$$

The resulting similarity scores form an activation map of dimensions $D \times H'' \times W''$, which indicates how strongly a prototypical feature appears in different parts of the image. As this map maintains the spatial structure D of the encoder output, it can be scaled or upsampled to match the input dimensions $D \times H' \times W'$ (9). The upsampled activation map can serve two purposes: first, as a heat map highlighting the regions in the input image that closely match the learned prototype, and second, to directly visualize the prototype itself.

To obtain a **single similarity score** for each prototype before the final classifier layer, in most of the prototype-based models, the corresponding activation map from the prototype layer $g(f(X))$ is passed through **global max pooling** layer. This score quantifies the presence of the prototypical part in any region of the input image. IAIA-BL uses top-k average pooling instead.

In PrototPViT (17), instead of a single similarity score per prototype, **summed similarity score** is calculated across sub-prototypes, and passed to the classifier layer. Furthermore, ProtoViT improves prototype matching with three key mechanisms:

- greedy matching, $\max_{\tilde{z}_f^i \in \tilde{z}_{A_j^k}} \cos(\tilde{z}_f^i, p_j^k)$, which pairs each sub-prototype with the most similar latent feature token using cosine similarity;
- adjacency mask A_j^k in greedy matching mechanism, to temporarily exclude feature tokens that are far from a selected sub-prototype or feature token pair;
- adaptive slots mechanism, $\tilde{\mathbb{1}}_{\{\text{include } p_j^k\}}$, which selects sub-prototypes based on their relevance and impact on performance.

$$g_{p_j}^{\text{greedy}}(\tilde{z}_f) = \frac{K}{\sum_{k=1}^{K} \tilde{\mathbb{1}}_{\{\text{include } p_j^k\}}} \sum_{k=1}^{K} \left\{ \max_{\tilde{z}_f^i \in \tilde{z}_{A_j^k}} \cos(\tilde{z}_f^i, p_j^k) \right\} \cdot \tilde{\mathbb{1}}_{\{\text{include } p_j^k\}}.$$

2.1.3 Classifier

Finally, M similarity scores are then passed to a classification layer h to compute the final predicted class probabilities for a given input image. Classifier h is often implemented as a single or two fully-connected layers.

2.2 Training algorithm

The training of the prototype-based interpretable models is generally divided into the following main stages.

2.2.1 Stochastic gradient descent (SGD) of layers before last layer

In the first stage, stochastic gradient descent (SGD) is applied to optimise the parameters of encoder $w_{encoder}$ and prototype layers $\mathbf{P} = \{p_j\}_{j=1}^{M}$, while keeping the classifier h fixed. This process encourages the feature space to structure itself so that image features from class k cluster around their corresponding prototypes p_j^k, while being well-separate from prototypes of other classes (9). The optimisation problem is formulated as:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{CE}}(h \circ g \circ f(X), Y) + \mathcal{L}_{\text{interp}}(f, g, h, X, Y, h \circ g \circ f(X)) \tag{2.1}$$

Here, the overall loss $\mathcal{L}_{\text{total}}$ is composed of two main components: $\mathcal{L}_{\text{CE}}$, the standard cross-entropy loss that measures the difference between the predicted class probabilities and the ground-truth labels Y, and $\mathcal{L}_{\text{interp}}$, a weighted sum of additional loss terms designed to encourage interpretability in the model's behavior (22).

Table 2.2: Loss functions used in prototype-based models.

Loss	**IAIA-BL**(11)	**ProtoPNet**(9)	**TesNet**(19)	**Def.-P.**(10)	**ProtoViT**(17)
Cross-entropy	Yes	Yes	Yes	Yes	Yes
Cluster	Yes	Yes	Yes	Yes	Yes
Separation	Yes	Yes	Yes	Yes	Yes
Orthogonality	No	No	Yes	Yes	Yes
Fine-Loss	Yes	No	No	No	No
Coherence	No	No	No	No	Yes
Subspace-sep.	No	No	Yes	No	No

The commonly used interpretability losses are the following.

- **Cluster loss** $\mathcal{L}_{\text{clust}}$ computes the average L^2 distance between the closest feature patch $\tilde{z}$ of an input image $f(x_i)$ and the prototypes p_j belonging to its ground-truth class $\mathbf{P}_{y_i}$. The minimization of this loss encourages the model to align parts of the input image with at least one prototype of its class (9).

$$\mathcal{L}_{\text{clust}} = \frac{1}{n} \sum_{i=1}^{n} \min_{j: p_j \in \mathbf{P}_{y_i}} \min_{\tilde{z} \in \text{patches}(f(x_i))} \|\tilde{z} - p_j\|_2^2 \tag{2.2}$$

- **Separation loss**: The goal of minimizing $\mathcal{L}_{\text{sep}}$ is to ensure that latent feature patches from a training image remain distant from prototypes that belong to different classes (9). It computes the average L^2 distance between the closest latent feature patch $\tilde{z}$ of an input image $f(x_i)$ and the prototypes p_j that do not belong to the same class as the training image.

$$\mathcal{L}_{\text{sep}} = -\frac{1}{n}\sum_{i=1}^{n} \min_{j: p_j \notin \mathbf{P}_{y_i}} \min_{\tilde{z} \in \text{patches}(f(x_i))} \|\tilde{z} - p_j\|_2^2 \tag{2.3}$$

- **Orthogonality loss**: for each class l with ρ assigned prototypes, $\mathbf{P}^{(l)} \in \mathbb{R}^{\rho \times Kd}$ is a matrix containing the flattened prototypes of class l, and $\mathbf{I}_\rho$ is the identity matrix of size $\rho \times \rho$. The orthogonality loss $\mathcal{L}_{\text{orth}}$ penalizes redundancy between prototypes within the same class, encouraging each prototype to learn distinctive features (10, 17).

$$\mathcal{L}_{\text{orth}} = \sum_{l=1}^{C} \|\mathbf{P}^{(l)}\mathbf{P}^{(l)T} - \mathbf{I}_\rho\|_F^2 \tag{2.4}$$

- **Fine loss** $\mathcal{L}_{\text{fine}}$ penalizes prototype activations on irrelevant regions, first introduced in (11) for radiologist-annotated training images. Here, $g_{p_j}(f(x_i))$ computes the similarity map between the convolutional features $f(x_i)$ and the j-th prototype p_j, and $\text{Upsample}(g_{p_j}(f(x_i)))$ performs bilinear upsampling to match the dimensions of the fine-annotation mask.

$$\mathcal{L}_{\text{fine}} = \sum_{i \in \mathcal{D}'} \left(\sum_{j:\text{class}(p_j) = y_i^{\text{margin}}} \|\mathbf{m}_i \odot \text{Upsample}(g_{p_j}(f(x_i)))\|_2 + \sum_{j:\text{class}(p_j) \neq y_i^{\text{margin}}} \|g_{p_j}(f(x_i))\|_2 \right) \tag{2.5}$$

- **Coherence loss** $\mathcal{L}_{\text{coh}}$, introduced in (17), ensures that sub-prototypes collectively represent the same feature. It penalizes sub-prototypes that are the most dissimilar to others within the same prototype, using cosine similarity. The slots indicator function is added to prune semantically distant sub-prototypes.

$$\mathcal{L}_{\text{coh}} = \frac{1}{m}\sum_{j=1}^{m} \max_{p_j^k, p_j^s \in p_j; p_j^s \neq p_j^k} \left(1 - \cos(p_j^k, p_j^s)\right) \cdot \mathbb{1}_{\{\text{Include } p_j^k\}} \mathbb{1}_{\{\text{Include } p_j^s\}} \tag{2.6}$$

- **Subspace separation**: Projection metric is a subspace distance metric on Grassmann manifold. It is used in TesNet(19), where each image feature map $\tilde{z}$ is projected onto a transparent embedding space spanned by a set of **basis concepts**. Each class k is assigned a subspace containing M_k orthogonal basis concepts, denoted as $B^{(k)} = \{b_j^{(k)}\}_{j=1}^{M_k}$, ensuring disentangled and interpretable representations. Subspace distance between orthonormal basis matrix of class k_1 and class k_1 in a projection metric is formulated as:

$$d_{proj}(B^{(k_1)}, B^{(k_2)}) = \frac{1}{\sqrt{2}} \left\| B^{(k_1)\top} B^{(k_1)} - B^{(k_2)\top} B^{(k_2)} \right\|_F ,$$

where $\| \cdot \|_F$ denotes the matrix Frobenius norm. Subspace separation loss that maximizes d_{proj} among each pair of subspaces is formulated as:

$$\mathcal{L}_{\text{sub.sep}} = -\frac{1}{\sqrt{2}} \sum_{k_1=1}^{K-1} \sum_{k_2=k_1+1}^{K} \left\| B^{(k_1)\top} B^{(k_1)} - B^{(k_2)\top} B^{(k_2)} \right\|_F$$

2.2.2 Projection of prototypes

To represent prototypes as patches of the training images, each prototype p_j^k is projected or ”pushed” onto the closest latent training patch that belongs to the same class k (9). Formally, a j-th prototype of class k p_j^k is updated as follows:

$$p_j^k \leftarrow \arg\min_{\tilde{z} \in Z_j} d(\tilde{z}, p_j^k) = \arg\max_{\tilde{z} \in Z_j} sim(\tilde{z}, p_j^k), \tag{2.7}$$

where $Z_j = \{\tilde{z} : \tilde{z} \in \text{patches}(f(x_i)) \, \forall i \text{ s.t. } y_i = k\}$. Once a prototype p_j^k is mapped to a specific latent patch x_i, the corresponding patch in the original image space is determined by identifying the most highly activated region. This process involves passing x_i through the trained model, enlarging the activation map from $g(f(x_i))$, and selecting the smallest bounding box that covers all pixels with activation values above a certain threshold.

2.2.3 Optimisation of the last layer

After the projection step, a convex optimisation is applied to the final classifier layer h while keeping all other parameters fixed. This step promotes **sparsity** in the last layer weights W by penalizing the L_1 norm of weights $W_{b,l}$, which are initially set to -0.5 for classes b and l, where $l \neq b$. By minimizing this loss, the model is encouraged to rely only on positive evidence for classification (9). The loss function is given by:

$$\mathcal{L}_h = \mathcal{L}_{\text{CE}}(h \circ g \circ f(x_i), y_i) + \lambda \sum_{l=1}^{K} \sum_{b:b \neq l}^{K} \|W_{b,l}\|_1 .$$

2.3 Related work in acceleration

Despite the growing interest in prototype networks, there remains a significant gap in research focused on accelerating their training or improving computational efficiency. Most studies prioritize enhancing model accuracy, refining prototype representation, or incorporating more expressive architectures and loss functions—often at the expense of increased training complexity. While some efforts have optimised the interpretability of saliency-based models like Grad-CAM (7, 23), they focus solely on improving interpretability rather than efficiency. Other types of interpretable models have seen acceleration efforts, such as leveraging Mixed Integer Optimisation for tree-based clustering algorithms (24), extending the CART algorithm to efficiently adapt to additive structures while maintaining interpretability (25), and IA-RED2, which reduces vision transformer computation by dynamically removing uncorrelated patches (26). However, these approaches primarily focus on interpretability through indirect means, such as heuristic rules or basic heat maps, and lack the structured, case-based reasoning and explicit decision-making process provided by prototype-based models.

Additionally, general coding-level acceleration techniques, such as half-precision training (e.g., FP16, FP8), model quantization, pruning, low-rank approximations, and knowledge distillation, have been explored to improve efficiency across various deep learning architectures, and can be potentially adapted to prototype-based models. However, these methods primarily focus on reducing computational cost at the hardware or numerical level rather than addressing the algorithmic inefficiencies specific to prototype-based models.

Overall, there is currently little to no work on accelerating these prototype-based models at a fundamental algorithmic level, independent of the backbone architecture. Thus, this work focuses on accelerating the training process specifically by simplifying the loss calculation, making prototype-based models more efficient while maintaining their accuracy and interpretability.

3 Acceleration

As shown in Table 2.2, the prototype-based models differ in their use of interpretability loss functions. However, the cluster and separation losses are the most commonly used across the majority of prototype-based models. These losses typically have the greatest influence or weight in the final $\mathcal{L}_{\text{total}}$ after $\mathcal{L}_{\text{CE}}$ under the default training configurations of the models. Traditionally, these models optimise cluster and separation costs using explicit min-max operations across feature patches and prototypes. However, this work proposes a simple modification: **replacing the explicit min-max distance computations with a global mean distance loss**. This modification moderately reduces per-epoch training time while maintaining competitive classification accuracy (within 1-2% of the original method).

The intuition behind the original cluster and separation loss functions, and a pseudo-code of their implementation:

- **Cluster loss** $\mathcal{L}_{\text{clust}}$ 2.2 encourages at least one feature patch of an input image to be close to a prototype belonging to its ground-truth class. This was computed by minimizing the smallest L_2 distance to a prototype of the correct class.
- **Separation loss** $\mathcal{L}_{\text{sep}}$ 2.3 ensures that no feature patch is close to prototypes of incorrect classes. This was done by maximizing the smallest L_2 distance to the incorrect class.

Algorithm 1 Original Cluster and Separation Loss Calculation

```
1: Input: dist, p_class_identity, label, max_dist
2: Output: $\mathcal{L}_{\text{clust}}, \mathcal{L}_{\text{sep}}$
3: p_of_correct_class ← p_class_identity[:, label]$^\top$
4: p_of_wrong_class ← 1 − p_of_correct_class
5: inverted_dist_correct ← max((max_dist − dist) · p_of_correct_class)
6: $\mathcal{L}_{\text{clust}}$ ← mean(max_dist − inverted_dist_correct)
7: inverted_dist_wrong ← max((max_dist − dist) · p_of_wrong_class)
8: $\mathcal{L}_{\text{sep}}$ ← −mean(max_dist − inverted_dist_wrong)
9: return $\mathcal{L}_{\text{clust}}, \mathcal{L}_{\text{sep}}$
```

In this work, the simplification is achieved by replacing the original cluster and separation loss with a simple mean of distances. This change eliminates the need for explicit min-max operations over prototypes and allows the model to optimise distances in a smoother, more stable manner.

Algorithm 2 Simplified Loss Calculation

```
1: Input: dist
2: Output: $\mathcal{L}_{\text{new}}$
3: $\mathcal{L}_{\text{new}} \leftarrow$ mean(dist)
4: return $\mathcal{L}_{\text{new}}$
```

This simplification doesn't affect the classification accuracy and quality of the learned prototypes for the following reasons.

- **Implicit approximation of cluster and separation losses**: Instead of explicitly minimizing the distance of the closest patch to a prototype, taking the mean of all distances and minimizing the loss of it encourages all patches to be closer to their respective nearest prototypes. This makes **every patch contribute to the loss** function instead of just the closest one, and makes the model learn to align features globally, not just one patch per class.
- **Smooth optimisation landscape** The original cluster and separation losses rely on max-min operations: max() for class-based separation finding farthest incorrect prototype, min() for class-based clustering finding closest correct prototype. These operations introduce sharp decision boundaries, making training more sensitive to outliers and potentially slower to converge. Proposed mean-based loss provides a **smoother gradient landscape**, allowing the model to converge faster and more stably.

This simple yet effective approach of acceleration provides a more efficient alternative to traditional prototype loss formulations while preserving their core intuition.

4

Experiments and Evaluation

4.1 Experimental setup

To assess the effectiveness of the proposed acceleration, execution time and accuracy were compared between the original and accelerated loss calculation implementations in ProtoPNet across different batch sizes. ProtoPNet was selected as it represents the standard and least computationally expensive implementation among prototype-based models. Notably, in all open-source prototype-based models (10, 11, 19, 17), cluster and separation cost calculations are same, with variations occurring in the distance or similarity computation methods.

Each training was conducted on a single NVIDIA Tesla T4 GPU with 16GB RAM. Execution time per batch iteration was measured using PyTorch CPU and CUDA Profiler (27). Execution time per epoch iteration was measured using Python's `time.perf_counter()`.

Models were trained and evaluated on the CUB-200-2011 dataset (28) of 200 bird species. Images were augmented and cropped as recommended in the official repository of the ProtoPNet's implementation. Each experiment ran for 12 epochs with the default hyperparameter configurations.

4.2 Execution time analysis

Table 4.1 presents the execution time and accuracy per batch and per epoch for different batch sizes in ProtoPNet with the original and accelerated loss calculation.

The original model exhibits significant execution time growth as the batch size increases. This trend is evident in both CPU and CUDA execution times. On the CPU, the per-batch time

Table 4.1: Comparison of execution time per batch iteration (in milliseconds), per epoch (in seconds) and accuracy of ProtoPNet with original and accelerated loss calculations on different batch sizes. ResNet-34 was used as the encoder backbone architecture.

Batch size	**Original**			**Accelerated**		
	p.batch(ms) CPU / CUDA	p.epoch(s)	Accuracy	p.batch(ms) CPU / CUDA	p.epoch(s)	Accuracy
40	210.73 / 85.32	983±2	77.6±0.1	202.13 / 75.6	932±3	76.8±0.1
80	386.87 / 154.47	903±3	78.1±0.1	363.29 / 128.53	829±3	76.1±0.1
160	1595.3 / 559.45	1720±50	75.4±0.1	713.38 / 255.89	814±4	74.6±0.2
200	3125.5 / 992.58	2610±50	75.3±0.3	898.54 / 320.65	818±2	74.7±0.1

increases from 210.73ms at batch size 40 to 3125.5ms at batch size 200, representing a 14.8× increase. Similarly, CUDA execution time per batch grows from 85.32ms to 992.58ms, showing an 11.6× increase. Per-epoch execution time follows a similar pattern, rising from 983s to 2610s (2.7× increase) on the original model.

This non-linear growth can be attributed to the computational complexity of the original loss calculations, which involve:

- **Per-class filtering:** The cluster and separation loss functions require masking prototypes based on class identity. As the batch size increases, the number of feature patches that need to be compared against prototypes grows proportionally. This leads to a substantial increase in the number of element-wise operations required to mask prototypes belonging to the correct and incorrect classes. Since this filtering operation is performed across all images in the batch, its computational cost scales with $O(B \times M)$, where B is the batch size and M is the number of prototypes. At large batch sizes, this results in a significant memory and compute overhead, causing execution time to increase non-linearly.

- **Min-max operations:** The loss calculation requires identifying the closest and farthest feature patches relative to each prototype. These min and max operations must be performed across all feature patches in the batch, resulting in costly reductions over high-dimensional tensors. As batch size increases, the number of reduction operations scales accordingly, leading to increased computational latency and reduced parallel efficiency on the GPU.

- **Memory overhead:** The original loss function requires storing and processing intermediate distance matrices of shape $B \times P \times H \times W$, where H and W are the spatial dimensions of the feature maps. As batch size increases, memory consumption grows quadratically, leading to higher VRAM usage and potential memory swapping. This can result in degraded GPU performance due to frequent memory accesses and cache thrashing.

These factors cause loss computation to grow with complexity $O(B \times P \times H \times W)$, where B is batch size, P is the number of prototypes, and H, W are spatial dimensions.

In contrast, the accelerated model demonstrates **improved scalability** with batch size. Specifically:

- Per-epoch execution time remains within a stable range of 800–930 seconds, even as batch size increases, indicating reduced computational overhead.
- Per-batch execution time scales in a near-linear fashion, avoiding the exponential growth observed in the original model.

4.3 Classification accuracy analysis

Despite the simplification, classification accuracy remains **within 1-2% of the original model** across all batch sizes. The results indicate:

- The accelerated model still effectively aligns input features with learned prototypes.
- The removal of explicit min-max operations does not significantly degrade the model's ability to separate classes.

4.4 Experiments on different model architectures

Table 4.2: Comparison of execution time per batch iteration (in milliseconds), per epoch (in seconds), and accuracy of different model architectures with original and accelerated loss calculations using a batch size of 80. ResNet-34 was used as the encoder backbone for ProtoPNet and TesNet, while DeiT-Tiny was used for ProtoViT.

Model	**Original**			**Accelerated**		
	p.batch(ms) CPU/CUDA	p.epoch(s)	Accuracy	p.batch(ms) CPU/CUDA	p.epoch(s)	Accuracy
ProtoPNet$_{80}$	386.8/154.4	903±3	78.1±0.1	363.2/128.5	829±3	76.1±0.1
TesNet$_{80}$	423.9/165.3	985±2	81.3±0.1	416.5/164.2	967±3	80.5±0.1
ProtoViT$_{80}$	1237/1652	2752±5	85.4±0.1	1207/1611	2731±5	85.1±0.1

As seen in Table 4.2, the accelerated loss calculation consistently reduces execution time across all tested model architectures while maintaining comparable accuracy.

From the results, it is observed that despite using the same backbone architecture, TesNet and ProtoPNet show differences in both accuracy and execution time. TesNet tends to take slightly longer per iteration, likely due to its additional loss components such as orthogonality loss, subspace separation loss, and subspace representation constraints, which contribute to a more structured feature space.

Additionally, the results indicate that ProtoViT achieves significantly higher accuracy compared to convolution-based architectures. However, this improvement comes at the cost of substantially higher computation time, as transformers are generally more compute-intensive than CNNs. Furthermore, ProtoViT incorporates additional interpretability losses, such as coherence and orthogonality losses, which further increase computational cost.

It is also noticeable that while the accelerated loss calculation provides performance gains, the acceleration effect is less pronounced in more sophisticated models such as TesNet and ProtoViT. This is likely because their execution time is dominated by other computationally expensive factors, including additional loss calculations ($\mathcal{L}_{orth}, \mathcal{L}_{coh}, \mathcal{L}_{sub.sep}$), a more intricate prototype representation (in TesNet), and a more complex backbone (in ProtoViT). As a result, while acceleration improves efficiency, the overall impact is more significant in simpler architectures like ProtoPNet, where $\mathcal{L}_{clus}$ and $\mathcal{L}_{sep}$ calculation constitutes a larger proportion of total execution time.

Overall, these findings suggest that while the accelerated loss calculation provides notable speed-ups, the choice of model architecture significantly impacts both accuracy and computational cost. Transformer-based models like ProtoViT offer improved classification performance but require considerably more resources, whereas CNN-based models such as ProtoPNet and TesNet strike a balance between efficiency and interpretability.

4.5 Evaluation of the interpretability

The interpretability of the learned prototypes was evaluated using two interpretability metrics: consistency score and stability score introduced in EvalProtoPNet (29). These metrics quantitatively assess whether prototypes remain meaningful across different images and under perturbations, addressing common issues in prototype-based networks. Table 4.3 presents a comparison of stability and consistency scores for the original and accelerated models across different batch sizes.

Table 4.3: Comparison of stability and consistency of ProtoPNet with original and accelerated loss calculations on different batch sizes

Batch size	**Original**		**Accelerated**	
	Stability	Consistency	Stability	Consistency
40	82.31	9.25	80.93	9.50
80	84.24	6.15	75.67	14.1
160	85.62	11.10	66.99	35.0
200	81.22	16.15	66.54	37.4

4.5.1 Consistency score

The consistency score measures the degree to which a prototype corresponds to the same object part across different images. A high consistency score indicates that a prototype is reliably associated with a specific semantic region, such as the head or wings of a bird, rather than activating different regions in different images.

- The original model shows low consistency across all batch sizes, with values ranging from 9.25% to 16.15%. The accelerated model achieves a substantial improvement in consistency, especially at higher batch sizes, increasing from 9.50% at batch size 40 to 37.4% at batch size 200. The trend indicates that the **accelerated model learns more stable object-part alignments**, particularly as batch size increases.

4.5.2 Stability score

The stability score assesses whether a prototype remains associated with the same object part before and after an input perturbation. A high stability score indicates robustness, meaning that small changes in the input do not cause the prototype to activate vastly different regions.

- In the original model, stability remains high (above 80%) across all batch sizes, showing minimal impact from perturbations. In contrast, the accelerated model exhibits a drop in stability, particularly for larger batch sizes, decreasing from 80.93% at batch size 40 to 66.54% at batch size 200. This reduction in stability suggests that while the accelerated model improves prototype consistency, it may also make prototypes more sensitive to small input variations.

4.5.3 Interpretability vs. Robustness trade-off

The original model prioritizes stability at the cost of low consistency, meaning prototypes remain mapped to arbitrary regions in a robust but inconsistent manner. The accelerated model improves consistency by learning more structured prototype mappings but introduces greater sensitivity to perturbations, resulting in lower stability at high batch sizes.

4.5.4 Scalability with batch size

The accelerated model shows better scalability with batch size in terms of consistency, which suggests that larger batch training helps prototypes converge towards more meaningful object parts. However, its decreasing stability at large batch sizes suggests a potential area for further improvement, such as introducing regularization techniques to balance consistency and robustness.

4.6 Overview

Prototype consistency improves significantly with acceleration, especially at higher batch sizes (9.5% to 37.4%). However, prototype stability decreases, indicating greater sensitivity to input perturbations. Larger batch sizes enhance consistency but reduce stability, revealing an interpretability-robustness trade-off.

These findings suggest that the accelerated model improves prototype interpretability while maintaining acceptable stability, making it more explainable yet sensitive. Future work could explore optimizing the balance between consistency and stability to enhance interpretability without compromising robustness.

Conclusions

This work introduces an acceleration method for prototype-based interpretable vision models by simplifying the cluster and separation loss calculations. The proposed modification eliminates computationally expensive min-max operations and replaces them with a simple mean distance loss, significantly reducing training time while maintaining competitive classification performance. Experimental results demonstrate that this modification **reduces per-epoch execution time by a factor of** $1.05\times$ **to** $3.2\times$, making training more scalable without modifying model architecture. While this acceleration improves efficiency, it results in a **minor accuracy drop of 0.4% to 1.2%**, depending on the dataset and model configuration.

Evaluation of interpretability metrics reveals that the acceleration substantially **improves prototype consistency**, particularly at larger batch sizes, where the consistency score increases from **9.5% to 37.4%**. However, this improvement comes at a minor cost of **reduced stability**, as prototypes become slightly more sensitive to input perturbations. Despite this trade-off, the accelerated model remains interpretable and provides a more computationally feasible alternative to traditional prototype-based training.

Beyond its effectiveness in ProtoPNet, this acceleration is directly applicable to any prototype-based interpretable vision model that computes cluster and separation losses, including TesNet, Deformable ProtoPNet, PPN, ProtoViT, and IAI-ABL (19, 10, 9, 17, 11). By incorporating this approach, these models can benefit from faster training and improved scalability, making them more practical for large-scale datasets and real-world applications.

To further enhance the training of prototype-based models, future research can focus on **accelerating similarity or distance calculations** between prototypes and images, as these operations constitute a significant computational bottleneck during model training. In the scope of this study, various strategies in this direction were explored, including top-k approximation, sparse representation, random projection, and dimensionality reduction techniques. However, these approaches either led to a significant drop in accuracy or provided little to no improvement in efficiency.

Additionally, other advanced **numerical optimization techniques** could be investigated to further enhance both computational efficiency and model performance. These include half-precision training (FP16/AMP) to reduce memory usage and computation time, LoRA (Low-Rank Adaptation) for more efficient model updates, and adaptive prototype selection strategies to improve robustness and stability. Exploring these methods may lead to more scalable and efficient prototype-based learning frameworks.

By demonstrating a computationally efficient approach that maintains interpretability while reducing training overhead, this work provides a foundation for making interpretable deep learning models more scalable and applicable to a broader range of tasks.

Bibliography

(1) Yosinski J, Clune J, Fuchs T, and Lipson H. Understanding Neural Networks through Deep Visualization. *Deep Learning Workshop at the 32nd International Conference on Machine Learning (ICML).* 2015

(2) Simonyan K, Vedaldi A, and Zisserman A. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *Workshop at the 2nd International Conference on Learning Representations (ICLR Workshop).* 2014

(3) Oord A van den, Kalchbrenner N, and Kavukcuoglu K. Pixel Recurrent Neural Networks. *Proceedings of the 33rd International Conference on Machine Learning (ICML).* 2016 :1747–56

(4) Zeiler MD and Fergus R. Visualizing and Understanding Convolutional Networks. *Proceedings of the European Conference on Computer Vision (ECCV).* 2014 :818–33

(5) Simonyan K, Vedaldi A, and Zisserman A. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *Workshop at the 2nd International Conference on Learning Representations (ICLR Workshop).* 2014

(6) Smilkov D, Thorat N, Kim B, Viégas F, and Wattenberg M. SmoothGrad: Removing Noise by Adding Noise. arXiv preprint arXiv:1706.03825 2017

(7) Selvaraju RR, Cogswell M, Das A, Vedantam R, Parikh D, and Batra D. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *Proceedings of the IEEE International Conference on Computer Vision (ICCV).* 2017 Oct

(8) Kim B, Wattenberg M, Gilmer J, Cai C, Wexler J, Viegas F, et al. Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV). *International Conference on Machine Learning.* PMLR, 2018 :2668–77

(9) Chen C, Li O, Tao D, Barnett A, Rudin C, and Su JK. This Looks Like That: Deep Learning for Interpretable Image Recognition. Advances in Neural Information Processing Systems 2019; 32

(10) Donnelly J, Barnett AJ, and Chen C. Deformable ProtoPNet: An Interpretable Image Classifier Using Deformable Prototypes. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2022 :10265–75

(11) Barnett AJ, Schwartz FR, Tao C, et al. A Case-Based Interpretable Deep Learning Model for Classification of Mass Lesions in Digital Mammography. Nature Machine Intelligence 2021; 3:1061–70. DOI: `10.1038/s42256-021-00423-x`

(12) Nauta M, Bree RV, and Seifert C. Neural Prototype Trees for Interpretable Fine-Grained Image Recognition. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2021 :14933–43

(13) Ma C, Zhao B, Chen C, and Rudin C. This Looks Like Those: Illuminating Prototypical Concepts Using Multiple Visualizations. Advances in Neural Information Processing Systems 2024; 36

(14) Rymarczyk D, Struski Ł, Górszczak M, Lewandowska K, Tabor J, and Zieliński B. Interpretable Image Classification with Differentiable Prototypes Assignment. *European Conference on Computer Vision.* Springer, 2022 :351–68

(15) Rymarczyk D, Struski Ł, Tabor J, and Zieliński B. ProtoPShare: Prototype Sharing for Interpretable Image Classification and Similarity Discovery. arXiv preprint arXiv:2011.14340 2020

(16) Sacha M, Rymarczyk D, Struski Ł, Tabor J, and Zieliński B. ProtoSeg: Interpretable Semantic Segmentation with Prototypical Parts. *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision.* 2023 :1481–92

(17) Ma C, Donnelly J, Liu W, Vosoughi S, Rudin C, and Chen C. Interpretable Image Classification with Adaptive Prototype-based Vision Transformers. arXiv preprint arXiv:2410.20722 2024

(18) Xue M, Huang Q, Zhang H, Cheng L, Song J, Wu M, and Song M. ProtoPFormer: Concentrating on Prototypical Parts in Vision Transformers for Interpretable Image Recognition. arXiv preprint arXiv:2208.10431 2022

(19) Wang J, Liu H, Wang X, and Jing L. Interpretable Image Recognition by Constructing Transparent Embedding Space. *Proceedings of the IEEE/CVF International Conference on Computer Vision.* 2021 :895–904

(20) Wolf TN, Pölsterl S, and Wachinger C. Don't PANIC: Prototypical Additive Neural Network for Interpretable Classification of Alzheimer's Disease. *International Conference on Information Processing in Medical Imaging.* Springer, 2023 :82–94

(21) Leeuw den Bouter M de, Pardo JL, Geradts Z, and Worring M. Proto-Explorer: Interpretable Forensic Analysis of Deepfake Videos Using Prototype Exploration and Refinement. Information Visualization 2023

(22) Willard F, Moffett L, Mokel E, Donnelly J, Guo S, Yang J, Kim G, Barnett AJ, and Rudin C. This Looks Better than That: Better Interpretable Models with ProtoPNeXt. arXiv preprint arXiv:2406.14675v1 [cs.CV] 2024

(23) Zhang Y, Zhu Y, Liu J, Yu W, and Jiang C. An Interpretability Optimization Method for Deep Learning Networks Based on Grad-CAM. IEEE Internet of Things Journal 2025; 12:3961–70. DOI: `10.1109/JIOT.2024.3485765`

(24) Bertsimas D, Orfanoudaki A, and Wiberg H. Interpretable clustering: an optimization approach. Machine Learning 2021; 110:89–138. DOI: `10.1007/s10994-020-05896-2`

(25) Tan YS, Singh C, Nasseri K, Agarwal A, Duncan J, Ronen O, Epland M, Kornblith A, and Yu B. Fast Interpretable Greedy-Tree Sums. Proceedings of the National Academy of Sciences of the United States of America (PNAS) 2025; 122:e2310151122. DOI: `10.1073/pnas.2310151122`

(26) Pan B, Panda R, Jiang Y, Wang Z, Feris R, and Oliva A. IA-RED2: Interpretability-Aware Redundancy Reduction for Vision Transformers. *35th Conference on Neural Information Processing Systems (NeurIPS).* 1MIT CSAIL, 2MIT-IBM Watson AI Lab, 3UT Austin. 2021

(27) PyTorch Team. Profiling PyTorch Models with the PyTorch Profiler. `https://pytorch.org/tutorials/recipes/recipes/profiler_recipe.html`. Accessed: 2025-01-25. 2023

(28) Wah C, Branson S, Welinder P, Perona P, and Belongie S. The Caltech-UCSD Birds-200-2011 Dataset. Tech. rep. CNS-TR-2011-001. California Institute of Technology, 2011

(29) Huang Q, Xue M, Huang W, Zhang H, Song J, Jing Y, and Song M. Evaluation and Improvement of Interpretability for Self-Explainable Part-Prototype Networks. *2023 IEEE/CVF International Conference on Computer Vision (ICCV).* Los Alamitos, CA, USA: IEEE Computer Society, 2023 Oct :2011–20. DOI: `10.1109/ICCV51070.2023.00192`

Declaration

I hereby declare that the work presented in this thesis is my own unless otherwise stated. To the best of my knowledge, the work is original, and ideas developed in collaboration with others have been appropriately referenced.

I acknowledge the use of ChatGPT 4o (OpenAI, `https://chat.openai.com/`) for fixing grammatical errors, improving word choice, and correcting misused phrases. I confirm that no content generated by AI has been presented as my own work.