Dissertation
Towards understanding computer vision system

by

Peijie Chen

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
May 4, 2024

Keywords: Machine learning, Explainable ML, Multi-modeling, Adversarial robustness

Approved by

Anh Nguyen, Assistant Professor of Computer Science and Software Engineering
Saad Biaz, Professor of Computer Science and Software Engineering
Heaton Haynes, Assistant Professor of Computer Science and Software Engineering
Cheryl D Seals, Charles W. Barkley Professor of Computer Science and Software Engineering
Gerry Dozier, Charles D. McCrary Eminent Chair Professor of Computer Science and Software Engineering

Abstract

In the realm of machine learning and deep neural networks, despite the significant strides made across diverse applications, the understanding and interpretation of these models, particularly under adversarial conditions and in the context of large-scale, multimodal frameworks, remain areas ripe for exploration. This thesis undertakes a detailed exploration into the biases inherent in standard and adversarially trained convolutional neural networks (CNNs). Our findings reveal a pronounced texture bias in standard CNNs, whereas their adversarially trained counterparts predominantly leverage shape cues, a distinction underscored by cue conflict experiments [31]. Through a meticulous neuron-level analysis employing NetDissect [5], we observe a marked increase in the monotonicity of neurons within robust networks, suggesting a fundamental shift in their information processing characteristics.

Building on the insights gleaned from our investigation into CNNs, we delve into the domain of large-scale foundation models, where we introduce gScoreCAM [17], an innovative visualization technique designed to illuminate the focal points of interest within images for OpenAI's CLIP [80], marrying high performance with unparalleled computational efficiency, achieving a significant leap over existing methods by 8 to 10 times in speed. However, recognizing the limitations of post-hoc explanation methods in terms of their reliability and fidelity [57], we propose the Part-based Image Classifiers with an Explainable and Editable Language Bottleneck (PEEB) [76]. PEEB represents a paradigm shift towards self-explainable AI frameworks, offering a unique layer of user controllability that enhances trustworthiness by enabling intuitive and transparent explanations directly during inference. This innovation is supported by introducing the Bird-11K and Dog-160 datasets, specifically curated to advance the development of part-based self-explanatory models.

This thesis's contributions illuminate the underlying biases and operational nuances of adversarially trained CNNs while simultaneously pioneering advancements in the explainability and

interpretability of complex, large-scale models. By introducing PEEB, complemented by gScore-CAM, we unveil novel pathways to understanding these intricate systems and empower users with a measure of control and transparency previously unattainable, heralding a new era of trust and clarity in artificial intelligence.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Machine learning, particularly through the use of convolutional neural networks (CNNs), has achieved remarkable success across a wide range of applications. However, despite these advances, challenges remain, especially in understanding and interpreting the decisions made by these models. For example, while CNNs, such as AlexNet [52], ResNet-50 [40], and GoogLeNet [97], have shown strong performance in tasks like image classification, they exhibit a notable bias towards texture over shape [31], differing significantly from human visual perception which relies heavily on shape. Furthermore, these models are susceptible to adversarial examples [96], revealing a critical gap in their ability to generalize to out-of-distribution data.

Adversarial training has emerged as a promising approach to mitigate these vulnerabilities by enhancing model robustness [109]. However, the internal transformations within CNNs post-adversarial training, particularly in how they process and prioritize visual features, remain largely unexplored. Moreover, the advent of large-scale, multimodal neural networks, exemplified by OpenAI's CLIP [80], has introduced new complexities. While these foundation models unlock new possibilities, they amplify the need for explainability, as their "black-box" nature makes it difficult to ascertain the basis of their outputs.

This thesis introduces a multifaceted exploration into CNNs' intricacies, addressing both the biases present in image classification tasks and the quest for explainability in machine learning models. We first dissect the shape and simplicity biases of adversarially robust ImageNet-trained CNNs, uncovering the dynamics between texture and shape biases and their implications on model performance across various data distributions. This analysis sets the stage for a deeper understanding of model behavior under adversarial conditions and out-of-distribution scenarios.

Subsequently, we delve into the realm of foundation models, with a particular focus on CLIP. By developing gScoreCAM, an efficient attribution method, we illuminate the operational mechanisms of CLIP, offering insights into how it processes and prioritizes information within images. This contribution not only enhances our understanding of CLIP but also sets a new benchmark in object localization tasks, marrying accuracy with computational efficiency.

At the heart of this thesis is the proposal of a **P**art-based Image Classifier with an **E**xplainable and **E**ditable Language **B**ottleneck (here after PEEB). PEEB embodies a significant leap towards achieving self-explainability in machine learning models. By leveraging human-interpretable knowledge for decision-making and explanation, PEEB offers a level of transparency and control previously unseen. This framework not only provides explanations that are directly aligned with human cognition but also introduces a controllable layer, allowing users to influence model behavior in a predictable and understandable manner.

1.1 Motivations

This thesis is propelled by the goal of narrowing the cognitive divide between human perception and artificial intelligence, especially in object recognition. Convolutional Neural Networks (CNNs), despite achieving remarkable success in image classification, exhibit a pronounced preference for texture over shape. This preference diverges from human perceptual strategies and makes these models susceptible to adversarial attacks. This observation not only underscores a critical gap in machine vision but also emphasizes the necessity for models capable of generalizing in a manner analogous to human sight.

The exploration of adversarial training has shown potential in nudging models towards a perception more aligned with human understanding, revealing, in the process, the opaque nature of their decision-making. Delving into foundation models like CLIP has further highlighted the imperative for interpretability. Although strides have been made with tools designed to unpack the rationale behind model decisions, these efforts have not fully offered a comprehensive solution. This realization has led to the insight that substantial progress requires the development of inherently self-explanatory models capable of rendering their decision-making processes transparent in terms congruent with human reasoning.

PEEB stands as a manifestation of this insight, marking a shift towards models that intrinsically provide explanations understandable to humans. By embedding explainability at its core, PEEB aims to mitigate the vulnerabilities and biases inherent in conventional machine learning models, aspiring to initiate a new era of AI systems that are robust and capable of broad generalization and characterized by their transparency. Consequently, this thesis delineates a trajectory from pinpointing fundamental challenges in machine perception to the realization of AI that harmonizes with human intellect, signifying a move towards machines that serve as tools and collaborators.

1.2 Contributions

The contributions of this thesis are diverse and significant, addressing several key challenges in the field of machine learning, particularly in understanding and improving the functionality of convolutional neural networks (CNNs) and large-scale foundation models. This work not only deepens our understanding of the biases and operational mechanisms of these models but also introduces novel methodologies and frameworks to enhance their explainability and usability. The primary contributions are as follows:

1. **Systematic Analysis of Shape/Texture Bias in CNNs:** We conducted an extensive investigation into the shape and texture biases of adversarially trained CNNs using two prevalent datasets in ML interpretability—cue-conflict [32] and NetDissect [5]. This study spans three widely-used architectures: AlexNet, GoogLeNet, and ResNet-50, all trained on ImageNet. Our findings reveal that robust classifiers (R classifiers) exhibit a strong preference for shapes over textures, contrasting sharply with standard classifiers (S classifiers). Additionally, our research demonstrates that adversarial training enhances the ability of CNNs to generalize

from adversarial to natural images by modifying their internal representations towards simpler and more generalized feature detection.

2. **Development of gScoreCAM for Efficient Object Localization in CLIP:** We proposed gScoreCAM, an advanced visual attribution tool that significantly improves the efficiency and accuracy of object localization tasks in CLIP models. gScoreCAM outperforms existing methods by reducing the computational overhead by approximately eight times while maintaining or exceeding accuracy. This tool provides a more precise understanding of what specific aspects within an image CLIP focuses on, enhancing our ability to interpret the decisions of this foundation model across diverse scenarios.

3. **Creation of PEEB, a Novel Self-Explainable Framework:** Perhaps the most transformative contribution of this thesis is the development of PEEB (Part-based Image Classifiers with an Explainable and Editable Language Bottleneck). PEEB integrates human-readable explanations directly into the decision-making process of image classification models. This framework utilizes a unique combination of visual and linguistic cues to facilitate an intuitive understanding of model judgments. In addition, we developed two extensive datasets, Bird-11K and Dog-140, to support the training and evaluation of PEEB and other similar models, addressing the lack of large, diverse datasets for fine-grained image classification and vision-language tasks.

4. **Empirical Validation and Application of Models:** Our models demonstrate superior performance in several benchmarks and practical applications. PEEB, in particular, significantly outperforms existing CLIP-based classifiers by wide margins in zero-shot and generalized zero-shot settings across multiple datasets. This underscores the effectiveness of PEEB in real-world scenarios, providing a reliable and interpretable alternative to existing models, which often rely heavily on opaque mechanisms.

These contributions significantly advance our understanding of the limitations and potential of contemporary AI systems. By addressing both theoretical and practical aspects, this thesis paves the way for developing more robust, generalizable, and transparent models in artificial intelligence.

1.3 Organization

This thesis is organized as follows: Chapter 1 introduces the core challenges and motivations behind this work. Chapter 2 details our study on the biases of CNNs and their behavior under adversarial training. In Chapter 3, we explore the foundation model CLIP and introduce gScore-CAM. Chapter 4 is dedicated to PEEB, our proposed solution for enhancing explainability and controllability in machine learning. Finally, we conclude with

Chapter 2

Analysis of Shape/Texture Bias in CNNs

2.1 Inconsistency between CNN and Human Perception

Human visual perception is inherently biased towards recognizing shapes rather than textures, a trait that contributes significantly to our ability to navigate and understand diverse and unfamiliar environments [31]. This shape bias allows humans to perform well on out-of-distribution (OOD) data, where conditions or items differ markedly from those seen during learning. In contrast, convolutional neural networks (CNNs), despite their prowess in image classification tasks, primarily leverage texture information to make decisions [32]. Studies have shown that while CNNs can align with human object recognition in some aspects [82, 92, 20], they often fail to generalize effectively to new, unseen examples due to their texture bias.

The reliance on texture makes CNNs susceptible to adversarial examples—subtly modified images that deceive the network into incorrect predictions while appearing normal to human eyes [96, 64]. Adversarial training has emerged as a primary method to counteract this by encouraging networks to learn from adversarial examples, thus shifting their focus from texture to shape [59]. This training approach not only aims to improve model robustness but also aligns CNNs closer to human-like perception by reducing their texture bias.

Moreover, when real images are incorporated into the training process, some architectures have shown improved accuracy on standard test sets [109], suggesting that adversarial training could also enhance general performance. This leads us to examine whether networks trained to be robust against adversarial attacks develop a preference for shapes over textures and whether this shift influences their performance on OOD images. While research has indicated that CNNs trained to emphasize shapes can better handle image corruptions and perturbations [31, 41, 8], it remains unclear if this advantage extends to adversarially robust networks.

In this chapter, we delve into these questions by evaluating the performance of adversarially trained networks (R networks) against standard networks (S networks) on ImageNet, a large-scale dataset commonly used to train state-of-the-art CNNs [88]. We explore the internal mechanisms by which adversarial training influences network behavior, particularly focusing on the changes in neural representations that lead to a potential increase in shape bias. Our study employs three well-known convolutional architectures—AlexNet, GoogLeNet, and ResNet-50—and analyzes their responses using datasets designed for machine learning interpretability and neuroscience studies, such as cue-conflict, NetDissect, and ImageNet-C [32, 5].

Our findings reveal several key insights:

1. Adversarially robust classifiers preferentially process shapes rather than textures, displaying a strong shape bias in approximately 67% of decisions, compared to only 25% in standard classifiers.

2. This shape bias enables R classifiers to outperform their standard counterparts on images with distorted or absent textures, such as stylized or silhouetted images.

3. Adversarial training enhances robustness by modifying neural processing to filter out noise and simplify the input patterns networks focus on, making them less susceptible to deceptive adversarial inputs.

4. Surprisingly, neurons in robust networks do not exhibit a clear bias towards either shape or texture but act as generalists, capable of handling low-level features across different visual categories.

These insights not only highlight the potential of adversarial training to bridge the gap between human and machine vision but also set the stage for subsequent discussions on the implications for building more reliable and interpretable AI systems.

2.2 Experiment setup

Networks To understand the effects of adversarial training across a wide range of architectures, we compare each pair of S and R models while keeping their network architectures constant. That is, we conduct all experiments on two groups of classifiers: (a) standard AlexNet, GoogLeNet, & ResNet-50 (hereafter, ResNet) models pre-trained on the 1000-class 2012 ImageNet dataset; and (b) three adversarially-robust counterparts i.e. AlexNet-R, GoogLeNet-R, & ResNet-R which were trained via adversarial training (see below).

Training A standard classifier with parameters θ was trained to minimize the cross-entropy loss L over pairs of (training example x, ground-truth label y) drawn from the ImageNet training set $\mathcal{D}$:

$$\arg\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}}\Big[L(\theta, x, y)\Big] \tag{2.1}$$

On the other hand, we trained each R classifier via Madry et al. [59] adversarial training framework where each real example x is changed by a perturbation Δ:

$$\arg\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}}\Big[\max_{\Delta\in\mathcal{P}} L(\theta, x + \Delta, y)\Big] \tag{2.2}$$

and $\mathcal{P}$ is the perturbation range allowed within an L_2 norm.

Hyperparameters The S models were downloaded from PyTorch model zoo [79]. We adversarially trained all R models using the robustness library [25], using *the same* hyperparameters in [26, 90, 4] because these previous works have shown that adversarial training significantly changed the inner-workings of ImageNet CNNs—i.e. becoming a strong image prior with perceptually-aligned deep features. That is, adversarial examples were generated using Projected Gradient Descent (PGD) [59] with an L_2 norm constraint ϵ of 3, a step size of 0.5, and 7 PGD-attack steps. R models were trained using an SGD optimizer for 90 epochs with a momentum of 0.9, an initial learning rate of 0.1 (which is reduced 10 times every 30 epochs), a weight decay of 10^{-4}, and a batch size of 256 on 4 Tesla-V100 GPU's.

Table 2.1: Top-1 accuracy (%) on 50K-image ImageNet validation-set and PGD adversarial examples.

	AlexNet	AlexNet-R	GoogLeNet	GoogLeNet-R	ResNet	ResNet-R
ImageNet	**56.52**	39.83	**69.78**	43.57	**76.13**	57.90
Adversarial	0.18	**22.27**	0.08	**31.23**	0.35	**36.11**

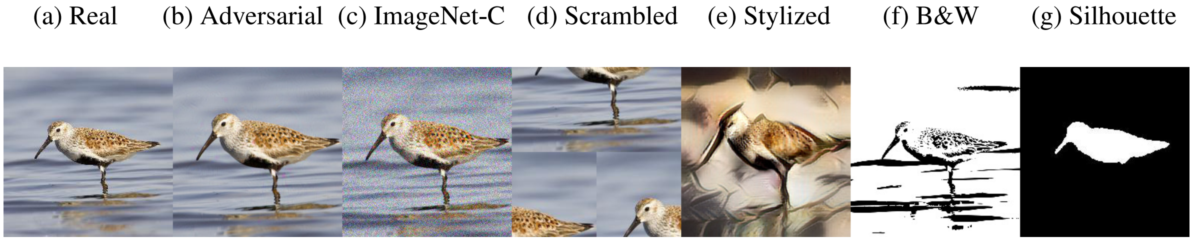

(a) Real (b) Adversarial (c) ImageNet-C (d) Scrambled (e) Stylized (f) B&W (g) Silhouette

Figure 2.1: Example distorted images (b–g). We show an example Gaussian-noise-added image (c) out of all 15 ImageNet-C types. See Fig. 6.5 for more examples.

Compared to the standard counterparts, R models have substantially higher adversarial accuracy but lower ImageNet validation-set accuracy (Table 2.1). To compute adversarial accuracy, we perturbed validation-set images with the same PGD attack settings as used in training.

Datasets To systematically analyze the behaviors of two types of networks—standard (S) and adversarially-robust (R)—with identical architectures but trained under different conditions, we utilized several datasets designed to probe different aspects of model performance and robustness.

The primary dataset used for benchmarking was the largest subset of the ImageNet validation set, referred to as ImageNet-CL. This subset contains images where both types of models achieve 100% accuracy, allowing for controlled comparisons. Specifically, for the AlexNet, GoogLeNet, and ResNet architectures, the subsets contain 17,693, 20,238, and 27,343 images, respectively [4].

In addition to ImageNet-CL, we further evaluated the networks on ImageNet-C, a dataset known for its rigorous assessment of model robustness against common types of image corruptions [41]. ImageNet-C includes fifteen distortions, such as noise, blur, and weather effects (Figure 2.1, column (c)). This dataset provides a robust platform to test how S and R models perform under adverse conditions, focusing on images that were originally correctly classified by both models in the ImageNet-CL dataset.

These evaluations help in understanding how networks perform on well-classified images and reveal how their behavior changes when encountering altered input features like textures or shapes, offering deeper insights into the fundamental differences between S and R network responses to varied visual stimuli.

2.3 Adversarial Training Reduces Texture Bias in ImageNet Classifiers

Understanding the decision-making basis of classifiers, whether they prioritize shape or texture, is crucial, particularly in adversarially robust networks trained to withstand deceptive inputs. Traditionally, ImageNet-trained CNNs are known to favor textures over shapes [32], a trait that adversarial training aims to modify.

To investigate the impact of adversarial training on this texture bias, we employed the cue-conflict dataset by Geirhos et al. [32], which contains images designed with conflicting texture and shape information. For instance, an image might display an elephant's texture mapped onto a cat's silhouette, challenging the model to decide whether to classify the image based on texture (elephant) or shape (cat).

Experimental Design Our analysis follows the experimental design set out by Geirhos et al. [32]. From the original dataset of 1,280 images, we removed 80 images lacking conflicting cues, thus focusing on 1,200 images with clear texture-shape conflicts. These images are categorized under 16 coarse labels from the MS COCO dataset [9], such as `cat` or `elephant`.

We evaluated both standard (S) and adversarially robust (R) models on these images. The models' output probability vectors for 1,000 classes were converted into 16-class vectors aligned with the MS COCO labels. We then assessed the models based on their correct classifications, determining whether they relied on shape or texture cues.

Findings on Shape and Texture Bias Our results significantly highlight the influence of adversarial training on model preference for shape recognition. Across three different CNN architectures—AlexNet, GoogLeNet, and ResNet-50—adversarially robust models demonstrated a preference for shapes in decision-making approximately 67.08% of the time. This marks a substantial increase compared to standard models, which exhibited a shape preference only 24.56% of the time (Table 2.2).

Table 2.2: While standard classifiers rely heavily on textures, R classifiers rely heavily on shapes. The top-1 accuracy scores (%) are computed on the cue-conflict dataset by Geirhos et al. [32].

	AlexNet		GoogLeNet		ResNet50			
	Standard	Robust	Standard	Robust	Standard	Robust	adv-prop PGD1	adv-prop PGD5
Texture	**73.61**	34.67	**74.91**	34.43	**77.79**	29.63	**68.24**	**63.11**
Shape	26.39	**65.32**	25.08	**65.56**	22.20	**70.36**	31.75	36.89

This significant shift in cognitive focus from textures to shapes, as induced by adversarial training, suggests not only an enhancement in robustness against adversarial attacks but also a fundamental alignment of CNNs' perceptual strategies with human visual processing, which predominantly relies on shape information. By reducing the texture bias by about 2.7 times, adversarially robust classifiers underscore the potential for creating more reliable and human-like artificial neural networks. This advancement paves the way for further investigations into how these perceptual adjustments can improve model performance and reliability across various challenging scenarios, including handling out-of-distribution data and responding to adversarial inputs.

2.4 Generalization Capabilities of Robust Networks

Adversarially trained networks (R models), known for their shape bias ([32]) as discussed in Section 2.3, provide an interesting case study for generalization to out-of-distribution (OOD) and distorted images. This analysis explores how these networks compare to standard ImageNet-trained networks (S models) when faced with various types of image distortions, especially focusing on their response to both distorted and shape-less images.

2.4.1 R Networks Show No Evidence of Better Generalization to Distorted Images

The ImageNet-C dataset, detailed in Section 2.2, contains a wide array of common image corruptions designed to test the robustness of models. It was observed that R models did not

Table 2.3: Adversarially-robust (R) models outperform vanilla (S) models when both are under PGD-adversarial attacks (b). While R models do not generalize well to common distortions (c), yet, they interestingly outperform S models on texture-less images (e–f). Here, we report top-1 accuracy scores (%) on the transformed images whose original real versions were correctly-labeled (a) by both S and R models. "ImageNet-C" column (c) shows the mean accuracy scores over all 15 distortion types. "Scrambled" column (d) shows the mean accuracy scores over three patch-scrambling types (details in Figure 2.3).

Network	(a)	(b)		Shape-less	Texture-less		
	Real	Adversarial	(c)	(d)	(e)	(f)	(g)
	ImageNet		ImageNet-C	Scrambled	Stylized	B&W	Silhouette
AlexNet	100	0.18	35.06	**34.59**	6.31	20.08	7.72
AlexNet-R	100	**22.27**	**44.94**	16.92	**9.11**	**35.25**	**9.30**
GoogLeNet	100	0.08	**58.27**	**49.74**	13.74	43.48	10.17
GoogLeNet-R	100	**31.23**	38.76	31.15	12.54	**44.55**	**24.12**
ResNet	100	0.35	**56.92**	**58.04**	10.68	16.96	3.95
ResNet-R	100	**36.11**	48.80	34.46	**15.62**	**53.89**	**22.30**

demonstrate an expected generalization advantage over S models on this dataset, as shown in Table 2.3. Despite the shape bias induced by adversarial training, the performance of R models was comparable to or worse than that of S models, indicating that a preference for shape does not inherently confer a broader generalization on all types of image corruptions.

2.5 Impact of Shape and Texture Biases on Network Performance

Our investigation into the performance differences between adversarially robust networks (R models) and standard ImageNet-trained networks (S models) on controlled image sets reveals significant insights into their operational biases and generalization capabilities.

Performance on Shape-less Images In experiments involving shape-less images, we modified ImageNet-CL images by scrambling them into grids of $p \times p$ patches (where $p \in \{2, 4, 8\}$, effectively removing coherent shape information while preserving texture details (as detailed in Appendix Section 6.1.1 and illustrated in Fig. 2.1d). The performance of R models was notably worse than that of S models on these manipulated images. Specifically, we observed a significant drop in accuracy for R models, ranging from 1.6 to 2.04 times lower than that for S models, as summarized in Table 2.3d. This substantial decrease highlights the reliance of R models on shape cues, underscoring a vulnerability when such cues are absent (see Fig. 2.2 for examples of network predictions on scrambled images).

Performance on Texture-less Images Conversely, in the assessment of texture-less images, where textures were either modified, reduced, or removed entirely, R models demonstrated superior performance. This set included stylized ImageNet images with altered textures, black-and-white images, and silhouettes—all of which minimize or eliminate texture cues (see Appendix Section 6.1.1 for details on image preparation methods). Here, R models consistently outperformed S models across all texture-less conditions, underscoring their capacity to utilize shape over texture. The results are detailed in Table 2.3e–g, indicating a robust generalization capability of R models in environments where shape information remains intact but textures are unreliable or misleading.

These contrasting outcomes from the experiments on shape-less versus texture-less images elucidate the distinct dependencies of adversarially trained models on shape features and their relative insensitivity to textures. Such findings emphasize the importance of considering these biases in the design and evaluation of neural networks, particularly for applications where robustness to image distortions or the ability to generalize across varied visual conditions is critical.

Table 2.4: Mean total variation (TV) of the conv layers of 6 models.

	AlexNet	AlexNet-R	GoogLeNet	GoogLeNet-R	ResNet	ResNet-R
Mean TV	110.20	**63.59**	36.53	**22.79**	18.35	19.96

2.6 What internal mechanisms make adversarially trained CNNs more robust than standard CNNs?

We have shown that after adversarial training, R models are more robust than S models on new adversarial examples generated for these pre-trained models via PGD attacks (Table 2.3b). Furthermore, on non-adversarial, high-frequency images, R models may also outperform S models (Table 6.1a; AlexNet-R) [112, 34].

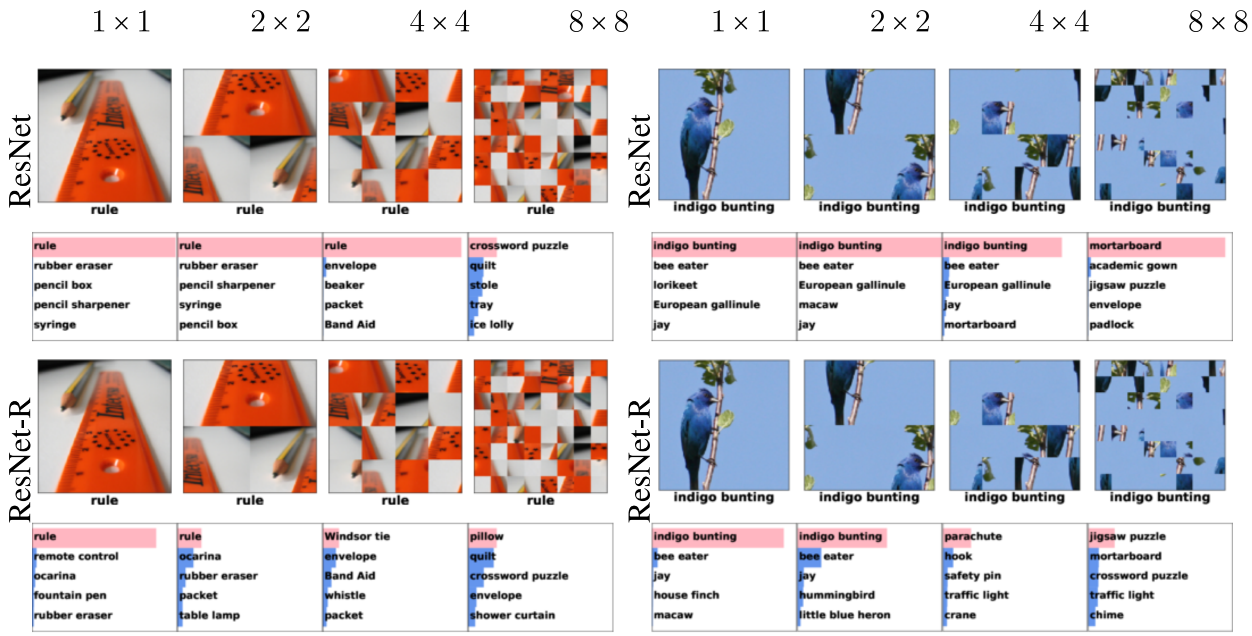


Figure 2.2: Qualitative examples showing the strong *texture* bias of standard CNNs (here, ResNet) and the strong *shape* bias of adversarially-robust models (here, ResNet-R). When patches are scrambled, ResNet-R confidence drops substantially and its top-1 predicted labels often change away from the original, correctly-predicted label, here, rule (left) and indigo bunting (right).

We further investigate the internal mechanisms that make R CNNs more robust to high-frequency noise by analyzing the networks at the weight (Section 2.6.1) and neuron (Section 2.6.2) levels.

2.6.1 Weight level: Smooth filters to block pixel-wise noise

Smoother filters To explain this phenomenon, we visualized the weights of all 64 conv1 filters (11×11×3), in both AlexNet and AlexNet-R, as RGB images. We compare each AlexNet conv1 filter with its nearest conv1 filter (via Spearman rank correlation) in AlexNet-R. Remarkably, R filters appear qualitatively much smoother than their counterparts (Figure 2.4a). The R filter bank is also less diverse, e.g. R edge detectors are often black-and-white in contrast to the colorful AlexNet edges (Figure 2.4b). A similar contrast was also seen for the GoogL-eNet and ResNet models (Figure 6.1).

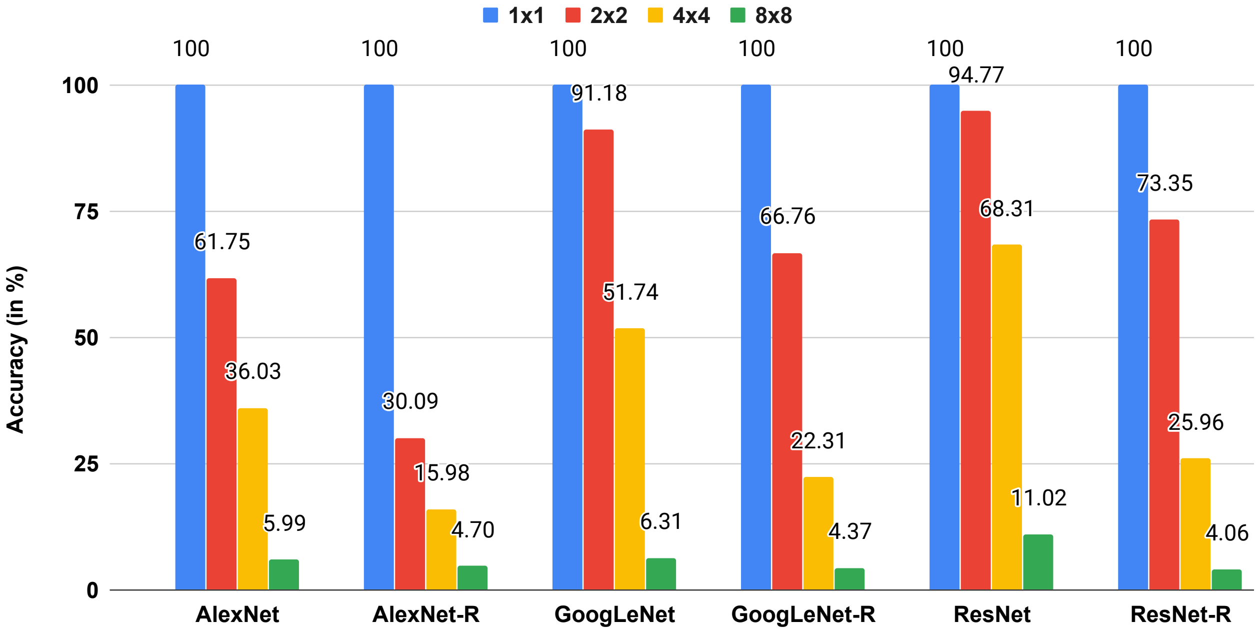
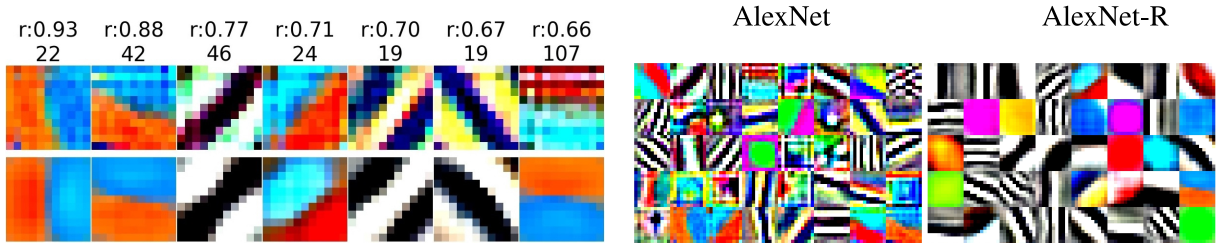


Figure 2.3: Standard CNNs substantially outperform R models on scrambled versions of the ImageNet-CL images due to their capability of recognizing images using textures. Here, we report top-1 accuracy (%) on the scrambled images whose original versions (ImageNet-CL) were correctly-labeled by both standard and R classifiers (hence, the 100% for 1×1 blue bars).

We also quantify the smoothness, in total variation (TV) [87], of the filters of all six models (Table 2.4) and found that, on average, the filters in R networks are much smoother.

For example, the mean TV of GoogLeNet-R is about 1.5 times smaller than GoogLeNet's. In almost all layers, R filters are smoother than S filters (Figure 6.9).

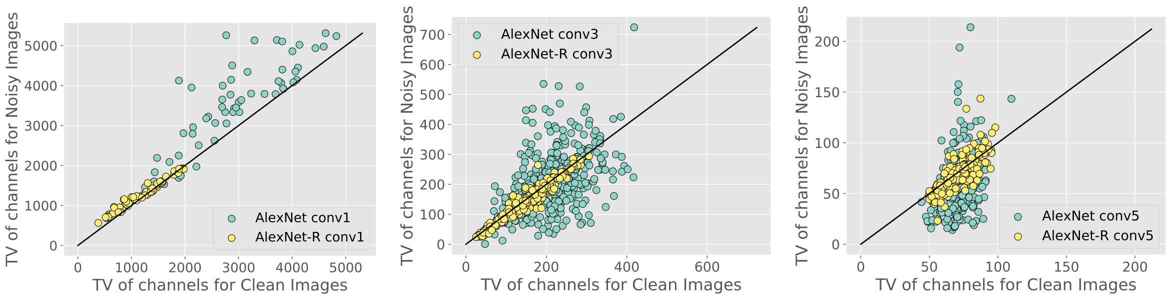
Blocking pixel-wise noise We hypothesize that the smoothness of filters makes R classifiers more robust against noisy images. To test this hypothesis, we computed the total variation of the channels across 5 conv layers when feeding ImageNet-CL images and their noisy versions (Fig. 2.1c; ImageNet-C Level 1 additive noise ~ $N(0, 0.08)$) to S and R models. At conv1, the smoothness of R activation maps remains almost unchanged before and after noise addition (Fig. 2.5a; yellow circles are on the diagonal line). In contrast, the conv1 filters in standard AlexNet allow Gaussian noise to pass through, yielding larger-TV channels (Fig. 2.5a; blue circles are mostly above the diagonal). That is, the smooth filters in **R models indeed can filter out pixel-wise Gaussian noise despite that R models were not explicitly trained on this image type**!



(a) Standard filters (top) & matching R filters (bottom) (b) 40 conv1 filters in AlexNet and AlexNet-R

Figure 2.4: **Left:** For each AlexNet conv1 filter (top row), we show the highest-correlated filter in AlexNet-R (bottom row), their Spearman rank correlation (e.g. r: 0.93) and the Total Variation (TV) difference (e.g. 22) between the top kernel and the bottom. Here, the TV differences are all positive i.e. AlexNet filters have higher TV. See Fig.6.2 for full plot. **Right:** conv1 filters of AlexNet-R are smoother and less diverse than the counterparts. See Figs. 6.1 for GoogLeNet & ResNet.

In higher layers, it is intuitive that the pixel-wise noise added to the input image might not necessarily cause activation maps in both S and R networks to be noisy because higher-layered units detect more abstract concepts. However, interestingly, we still found that R channels have consistently less mean TV (Fig. 2.5b–c). Our result suggests that most de-noising effects occur at lower layers (which contain more generic features) instead of higher layers (which contain more task-specific features).



(a) 64 conv1 channels (b) 384 conv3 channels (c) 256 conv5 channels

Figure 2.5: In each subpanel, one point shows the mean Total Variation (TV) of one channel when running clean ImageNet-CL images and their noisy versions through AlexNet (🔵) or AlexNet-R (🟡). R channels have similar TV before and after adding noise, suggesting that conv1 kernels filter out the added noise. In higher layers (conv3 and conv5), R channels are consistently more invariant to the input noise than S channels (🟡 dots are clustered around the diagonal line while 🔵 dots have higher variance).

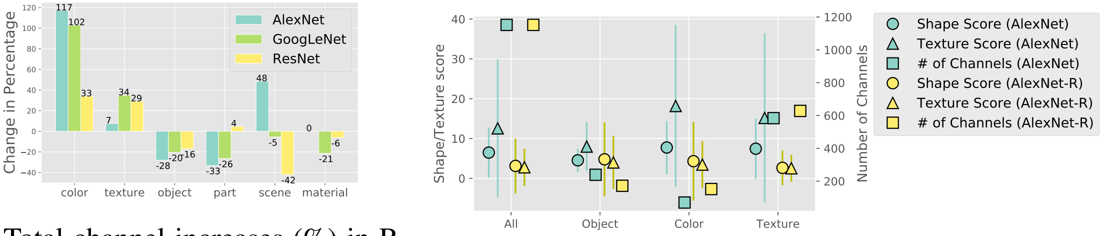
2.6.2 Neuron level: Robust neurons prefer lower-level and fewer inputs

Here, via the NetDissect framework, we wish to characterize how adversarial training changed the hidden neurons in R networks to make R classifiers more adversarially robust.

Network Dissection (hereafter, NetDissect) is a common framework for quantifying the functions of a neuron by computing the Intersection over Union (IoU) between each activation map (i.e. channels) and the human-annotated segmentation maps for the same input images.

That is, each channel is given an IoU score per human-defined concept (e.g. dog or zigzagged) indicating its accuracy in detecting images of that concept. A channel is tested for its accuracy on all ~1,400 concepts, which span across six coarse categories: object, part, scene, texture, color, and material [5] (c.f. Figure 6.11 for example NetDissect images in texture and color concepts).

Following [5], we assign each channel C a main functional label i.e. the concept that C has the highest IoU with. In both S and R models, we ran NetDissect on all 1152, 5808, and 3904 channels from, respectively, 5, 12, and 5 main convolutional layers (post-ReLU) of the AlexNet, GoogLeNet, and ResNet-50 architectures (c.f. Section 6.1.2 for more details of layers used).



(a) Total channel increases (%) in R models

(b) Shape & Texture score of AlexNet & AlexNet-R

Figure 2.6: **Left:** For all three architectures, the numbers of NetDissect color and texture detectors in R models increase, e.g. by 117% and 7%, respectively, for AlexNet, while the number of object units decreases by 28% (a). See Figure 6.3 for layer-wise plots for detectors in other categories. **Right:** The average Shape (●) and Texture (▲) scores over all channels in the entire network ("All") or in a NetDissect category ("Object", "Color", and "Texture"). While AlexNet-R has more color and texture channels (🟨 above 🟦), these R channels are not heavily shape- or texture-biased. In contrast, the corresponding channels in AlexNet are heavily texture-biased (▲ is almost 2× of ●).

Shift to detecting more low-level features i.e. colors and textures We find a consistent trend—adversarial training resulted in substantially more filters that detect colors and textures (i.e. in R models) in exchange for fewer object and part detectors.

For example, throughout the same GoogLeNet architecture, we observed a 102% and a 34% increase in color and texture detectors, respectively, in the R model, but 20% and 26% fewer object and part detectors, compared to the S model (c.f. Figure 2.6a). After adversarial training, ~11%, 15%, and 10% of all hidden neurons (in the tested layers) in AlexNet, GoogLeNet, and ResNet, respectively, shift their roles to detecting lower-level features (i.e. textures and colors) instead of higher-level features (see feature visualizations in Fig. 2.7).

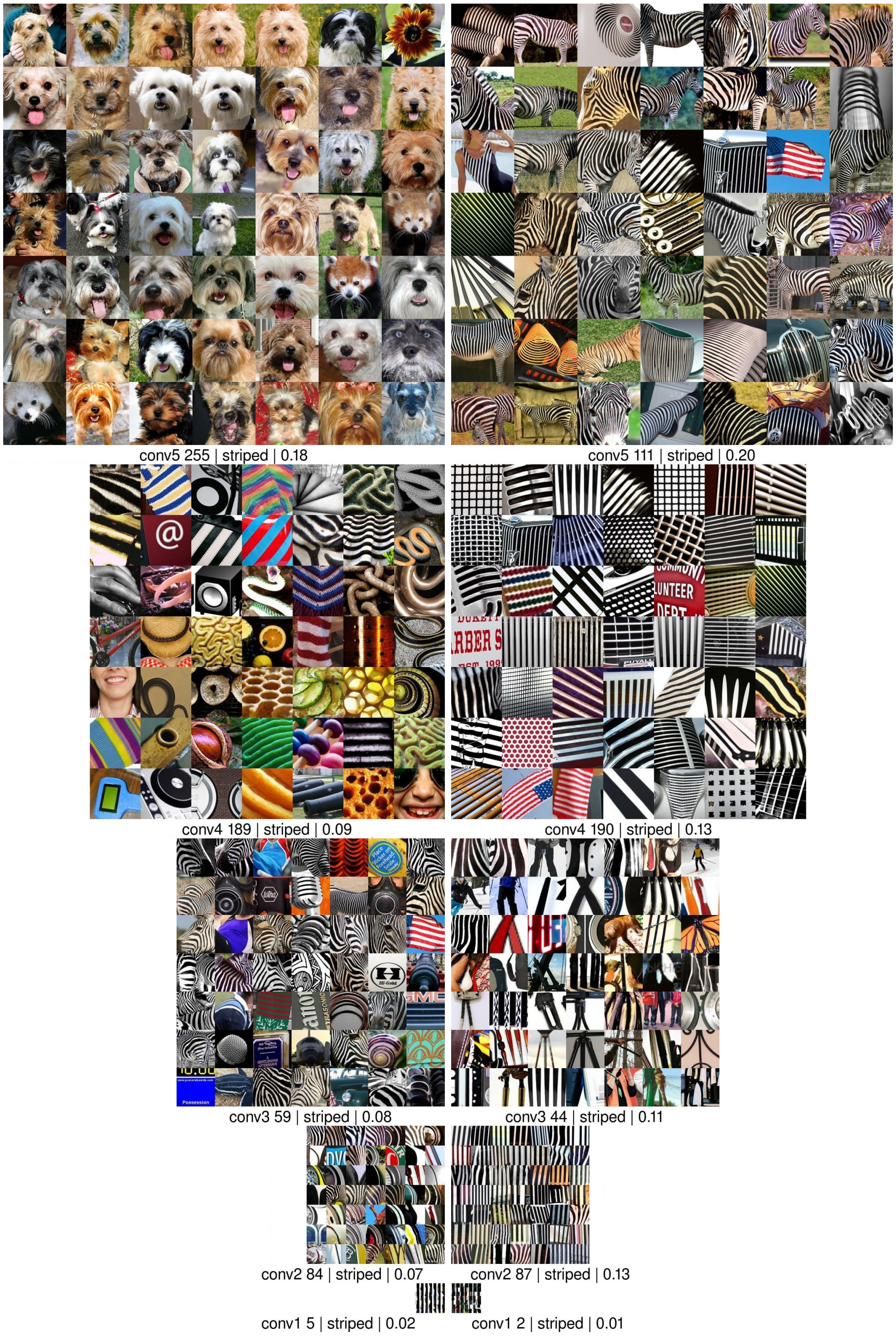
Across three architectures, the increases in texture and color channels are often larger in higher layers. The largest functional shifts in higher layers can be because the higher-layered units are more task-specific [65, 113].

Consistent findings with ResNet CNNs trained on Stylized ImageNet We also compare the shape-biased ResNet-R with ResNet-SIN, i.e. a ResNet-50 trained exclusively on stylized ImageNet images where textures are removed via stylization [32]. ResNet-SIN also has a strong shape bias of 81.37%.[1] Interestingly, similar to ResNet-R, ResNet-SIN also has more low-level feature detectors (colors and textures) and fewer high-level feature detectors (objects and parts) than the vanilla ResNet (Figure 2.8). In contrast, finetuning this ResNet-SIN on ImageNet remarkably changes the model to be *texture-biased* (at a 79.7% texture bias) and to contain fewer texture and more object and part units (Figure 2.8; ResNet-SIN+IN vs. ResNet-SIN).

That is, training or finetuning on ImageNet tend to cause CNNs to be more texture-biased and contain more high-level features (i.e. detecting objects and parts). In contrast, training on adversarial examples or texture-distorted images causes CNNs to focus more on shapes and learn more generic, low-level features.

Shift to detecting simpler objects Analyzing the concepts in the object category where we observed largest changes in channel count, we find evidence that neurons change from detecting complex to simpler objects. That is, for each NetDissect concept, we computed the difference in the numbers of channels between the S and R model. In the same object category, AlexNet-R model has substantially fewer channels detecting complex concepts e.g. −30 dog, −13 cat, and −11

[1] model_A in `https://github.com/rgeirhos/texture-vs-shape/`. See Table 4 in [32].


conv5 255 | striped | 0.18
conv5 111 | striped | 0.20
conv4 189 | striped | 0.09
conv4 190 | striped | 0.13
conv3 59 | striped | 0.08
conv3 44 | striped | 0.11
conv2 84 | striped | 0.07
conv2 87 | striped | 0.13
conv1 5 | striped | 0.02
conv1 2 | striped | 0.01

(a) AlexNet

(b) AlexNet-R

Figure 2.7: Each 7×7 grid shows the top-49 training-set images that highest activate the center unit in a channel. Each column shows five highest-IoU striped concept channels, each from one AlexNet's conv layer in their original resolutions. From top to bottom, AlexNet-R (b) consistently preferred striped patterns, i.e., edges (conv1), vertical bars (conv2), tools, to grids and zebra (conv5). In contrast, AlexNet striped images (a) are much more diverse, including curly patterns (conv4) and dog faces (conv5).

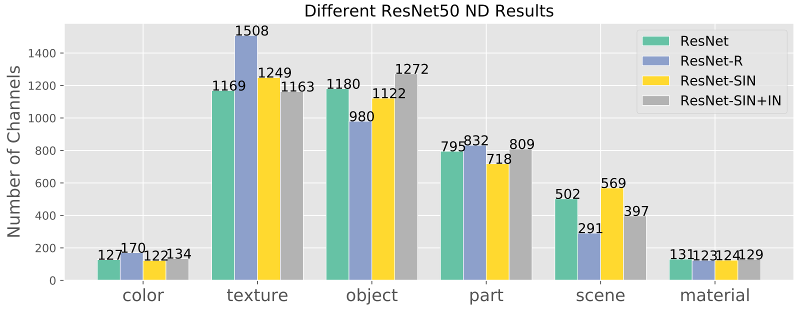
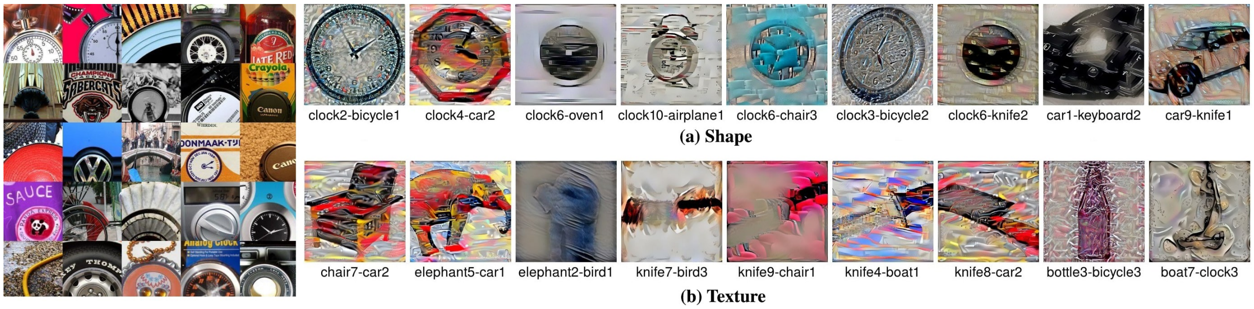


Figure 2.8: Each column shows the number of channels (in a ResNet-50 model) categorized into one of the NetDissect categories. For example, the standard ImageNet-trained ResNet has 127 color detectors (leftmost column). Here, we compare the neural functions among four different ResNet-50 models trained differently: (1) ResNet was trained on ImageNet; (2) ResNet-R was trained via PGD-adversarial training; (3) ResNet-SIN was trained on texture-removed ImageNet from [32]; and (4) ResNet-SIN+IN was a ResNet-SIN that was then finetuned on ImageNet. Training on texture-removed ImageNet or adversarial examples consistently produces CNNs that are (a) heavily shape-biased; (b) contain more low-level, texture features; (c) fewer high-level, object detectors compared to training on ImageNet which produces texture-biased CNNs (ResNet and ResNet-SIN+IN).



(a) Shape

(b) Texture

Figure 2.9: **Left:** Top-25 highest-activation images of the AlexNet unit conv4$_{19}$, which has a NetDissect label of spiralled under texture category. The unit prefers circular patterns, including car wheels and clocks. **Right:** Example *cue-conflict* images originally labeled by AlexNet as shape (top) or texture (bottom) but that were given a different label after the unit conv4$_{19}$ is ablated. For example, “clock2-bicycle1” is a cue-conflict image that has the shape of a clock and the texture of a bicycle. Qualitatively, the unit helps AlexNet detect clocks and cars using shapes (top) and reddish pink cars, birds, chairs, and bicycles using textures (bottom). The unit has Shape and Texture scores of 18 and 22, respectively.. See Figure 6.6 & Figure 6.7 for more examples.

person detectors (Figure 6.10b; rightmost columns), compared to the standard network. In contrast, the R model has more channels detecting simpler concepts, e.g. +40 sky and +12 ceiling channels (Figure 6.10b; leftmost columns). The top-49 images that highest-activated R units across five conv layers also show their strong preference for simpler backgrounds and objects (Figure 2.7).

Shift to detecting fewer unique concepts The previous sections have revealed that neurons in R models often prefer images that are pixel-wise smoother (Section 2.6.1) and of lower-level features (Section 2.6.2), compared to S neurons. Another important property of the complexity of the function computed at each neuron is the diversity of types of inputs detected by the neuron [66, 67]. Here, we compare the diversity score of NetDissect concepts detected by units in S and R networks. For each channel C, we calculated a diversity score, i.e. the number of unique concepts that C detects with an IoU score ≥ 0.01.

Interestingly, on average, an R unit fires for 1.16 times fewer unique concepts than an S unit (22.43 vs. 26.07; c.f. Figure 6.12a). Similar trends were observed in ResNet (Figure 6.12b). Qualitatively comparing the highest-activation training-set images by the highest-IoU channels in both networks, for the same most-frequent concepts (e.g. striped), often confirms a striking difference: R units prefer a less diverse set of inputs (Fig. 2.7). As R hidden units fire for fewer concepts, i.e. significantly fewer inputs, the space for adversarial inputs to cause R models to misbehave is strictly smaller.

2.6.3 Which neurons are important for shape-based or texture-based image classification?

To understand how the found changes in R neurons (Section 2.6) relate to the shape bias of R CNNs (Section 2.3), here, we zero out every channel, one at a time, in S and R networks and measure the performance drop in recognizing shapes and textures from cue-conflict images.

Shape & Texture scores For each channel, we computed a "Shape score", i.e. the number of images originally correctly labeled into the shape class by the network but that, after the ablation, are labeled differently (examples in Figure 2.9a–b). Similarly, we computed a Texture score per channel. The Shape and Texture scores quantify the importance of a channel in image classification using shapes and textures, respectively.

First, we find that the **channels labeled** texture **by NetDissect are not only important to texture-based but also shape-based classification**. That is, on average, zero-ing out these channels caused non-zero Texture and Shape scores (Figure 2.6b; Texture ○ and △ are above 0). See Figure 2.9 for an example of texture channels with high Shape and Texture scores.

This result is aligned with the fact that R networks consistently have more texture units (Figure 2.6a) but are shape-biased (Section 2.3).

Second, the texture units are, as expected, highly texture-biased in AlexNet (Figure 2.6b Texture; ▲ is almost 2× of ●). However, surprisingly, those texture **units in AlexNet-R are neither strongly shape-biased nor texture-biased** (Figure 2.6b; Texture ○ ≈ △). That is, across all three groups of the object, color, and texture, **R neurons appear mostly to be generalist, low-level feature detectors**. This generalist property might be why R networks are more effective in transfer learning than S networks [89].

Finally, the contrast above between the texture bias of S and R channels (Figure 2.6b) reminds researchers that the single semantic label assigned by NetDissect to each neuron is not describing a full picture of what the neuron does and how it helps in downstream tasks. To our knowledge, this is the first work to align the NetDissect and cue-conflict frameworks to study how individual neurons contribute to the generalizability and shape bias of the entire network.

2.7 Discussion and Conclusion

This chapter has detailed the differential reliance on shape and texture cues between standard and adversarially robust convolutional neural networks (CNNs). Unlike their standard counterparts (S networks), we discovered that R networks exhibit a pronounced preference for shape information. This finding underscores the potential to merge these models—integrating an R network's shape bias with an S network's texture sensitivity—to form a more robust and interpretable machine learning model. Such a hybrid approach could enhance generalizability across out-of-distribution data and provide clearer insights into the feature dependencies that underlie model predictions.

Moreover, our analysis revealed that units traditionally identified as texture detectors contribute equally to shape-based recognition tasks. This observation challenges the notion that texture detection is solely pivotal for texture-biased recognition and suggests a need to reevaluate how features are categorized in network interpretability studies. Adjustments might include incorporating low-frequency patterns like single lines or silhouettes to better capture the nuanced roles these units play.

In light of these findings, we propose that future efforts in developing CNNs, particularly those aimed at robust performance across varied visual tasks, should not only focus on modifying internal network mechanisms. Instead, there is significant value in designing systems that allow end users to exert higher-level control over model behavior. By enabling users to specify or adjust the balance between shape and texture processing, models can be tailored to more closely align with human visual processing and to meet specific task requirements.

Our exploration into the inherent capabilities and biases of CNNs thus not only enriches our understanding of these complex systems but paves the way for the next generation of interpretable and adaptable artificial intelligence tools. The next chapter will further explore the realm of foundation models such as CLIP [80], focusing on understanding their underlying mechanisms and their implications for advancing machine learning towards more human-like processing.

Chapter 3

gScoreCAM: Understand the foundation model CLIP

3.1 The Emergence of Foundation Models

Foundation models, large-scale, multimodal neural networks trained on extensive web data, have quickly become indispensable in various domains ranging from academic research to industry applications [7]. OpenAI's CLIP [80], a prominent example of such models, has demonstrated remarkable capabilities by learning to match captions with images. Within a year of its release, it has been leveraged in diverse applications including text-to-image synthesis [73, 63, 48], video retrieval [58, 54], visual question answering [94], and image editing [53, 103]. Despite their growing ubiquity and scaling, the internal mechanisms of foundation models like CLIP remain largely enigmatic, which poses risks related to safe deployment and bias [93, 102, 55].

Understanding these models is crucial not only to mitigate potential harms but also to enhance their functionality [77]. For instance, it is particularly intriguing that a model as sophisticated as CLIP can be misled by simple textual cues on objects [35]. This calls for robust methods to interpret and verify the model's decision-making processes, especially in complex scenes where the model's focus and the rationale behind its predictions can be ambiguous.

Current methods for visualizing and interpreting CLIP and other vision-language models predominantly include ViT-based techniques which illuminate similarities between image and text tokens [13, 95, 1]. However, these methods often fall short in complex, multi-object settings or do not apply to all model architectures since many rely on cross-attention mechanisms absent in CLIP [49, 80]. Alternatively, methods developed for convolutional networks like various CAM-based

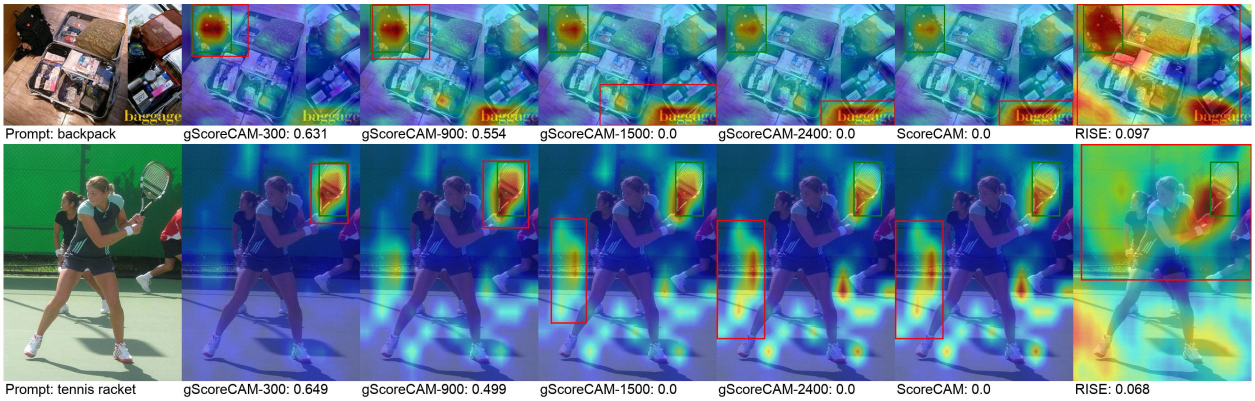


Figure 3.1: In a complex MS COCO scene, SotA feature importance methods often produce noisy heatmaps for CLIP RN50x16, questioning what objects are the most important to CLIP. Here, RISE [75] heatmap covers both suitcases and the text "baggage" (top row) and ScoreCAM [105] heatmap highlights both the racket and the background (bottom row), yielding a low IoU of 0.0 w.r.t. the groundtruth box (□). By using only the top-k channels of the highest gradients, we (1) localize *the most important* objects in a complex scene (e.g., here, racket at IoU of 0.649); and (2) produce a SotA zero-shot, open-vocabulary, object localization method for MS COCO and PartImageNet.

approaches [117, 91, 11] tend to produce noisy or inaccurate heatmaps when applied to CLIP's ResNet variants [35].

To address these limitations, we introduce gScoreCAM, a novel interpretability method designed specifically for CLIP. gScoreCAM significantly reduces the computational complexity of existing ScoreCAM [105] approaches by focusing only on the most gradient-significant channels of the model. This method not only accelerates the interpretability process by up to 10 times but also enhances the accuracy of zero-shot, open-vocabulary object localization in challenging datasets like MS COCO and PartImageNet [56, 39]. Through gScoreCAM, we aim to provide a more practical and precise tool for discerning the most crucial elements in images that influence CLIP's decision-making, thereby offering deeper insights into the inner workings of foundation models.

3.2 gScoreCAM: A Simple Tool To Visualize CLIP

We first describe the original Class Activation Map (CAM) method [117] and then ScoreCAM [105] before introducing our gScoreCAM, which extends ScoreCAM and is the state-of-the-art method for CLIP ResNets in zero-shot, open-vocab localization.

3.2.1 Revisiting CAM and ScoreCAM

CAM [117] The idea of CAM is to use the output of global average pooling in CNNs to generate a class activation map that indicates a region in the image for a given class. For a given CNN, CAM takes the activation maps from the last convolutional layer and calculates a weighted sum of the activation maps as the class activation map. The class activation map M^c_{CAM} is given by:

$$M^c_{CAM} = \sum_i w_i A^c_i \tag{3.1}$$

where w_i is the i^{th} weight of global average pooling layer, A_i is the i^{th} feature map (output of the last convolutional layer). $i \in C$, C is the number of channels in the last convolutional layer.

ScoreCAM [105] ScoreCAM utilizes the network itself to derive the weights for the activation maps so that it no longer relies on the weights in the average pooling layer. ScoreCAM algorithm is given by Algorithm 1.

Algorithm 1 ScoreCAM Algorithm for Visualizing Convolutional Neural Networks

```
Input: Input image $I \in \mathbb{R}^{w \times h}$, target class $c$, CNN model $F$, target layer $T$
Output: A heatmap $M \in \mathbb{R}^{u \times v}$
Extract activation maps $\{A_i\}_{i=1}^{C}$ from layer $T$ of $F$ using $I$.
for $i = 1$ to $C$ do
    Up-sample activation map $A_i$ to the size of $I$ using bilinear interpolation to obtain mask $M_i^{mask}$.
end
Initialize a vector $S^c \in \mathbb{R}^C$ to store scores for each mask.
for $i = 1$ to $C$ do
    Apply mask $M_i^{mask}$ to $I$ to produce a masked image $I_i^{mask} = I \odot M_i^{mask}$.
    Compute the prediction score $s_i^c = F(I_i^{mask})^c$ for the target class $c$.
    Store the score $s_i^c$ in $S^c$.
end
Compute weights $W = \text{softmax}(S^c)$.
Compute the ScoreCAM heatmap $M = \sum_{i=1}^{C} W_i \cdot A_i$.
```

Algorithm 2 gScoreCAM Algorithm for Visualizing CLIP ResNets

```
Input: Input image $I \in \mathbb{R}^{w \times h}$, a prompt $P$, target layer $T$
Output: A heatmap $M \in \mathbb{R}^{u \times v}$
Run one forward pass through CLIP $(I, P)$ to get 3,072 channel activations $\{A_i\}_{3072}$ at layer $T$.
Run one backward pass to get the channel-wise gradients.
Select the top-$k$ channels based on the highest gradients and use them as masks to generate $k$
  masked input images, $\{I_i^*\}_k$.
for $i = 1$ to $k$ do
    Run a forward pass through CLIP $(I_i^*, P)$ to get a cosine similarity score for each masked image
      $I_i^*$.
end
Use the cosine similarity scores as weights to linearly combine the $k$ channels into a single
  gScoreCAM heatmap.
```

3.2.2 Proposed method: Gradient-guided ScoreCAM (gScoreCAM)

ScoreCAM requires computing all weights w^c through CNN, resulting in high computational overhead. We propose a simple but much more efficient modification based on ScoreCAM. Algorithm 2 shows a simplified workflow of our method.

Instead of exhaustively computing all the weights, we only compute a small subset of w^c (via CNN) and assign others to zero. The idea is that only a small subset of the channels is important for the final class activation map. We use the gradient of the target layer to help us select the important channels. First, we define a function to derive the first k index from a sequence:

Definition 1 (sort) *Let A be a sequence of real numbers. The sequence $sort(A) \subset \mathbb{R}$ is defined as the result of sorting A in descending order, preserving duplicate elements.*

Definition 2 (topIndex) *Let $N \in \mathbb{N}$ and for all $0 \leq i \leq N$, let $a_i \in \mathbb{R}$ and let $A = (a_i)_{i=0}^{N}$ be a sequence. Define $B = (b_i)_{i=0}^{N} = sort(A)$. Then $topIndex(A, k)$ is defined as the set of all indices c such that $a_c > b_k$ and $c \in \mathbb{Z}$.*

Definition 3 (gScoreCAM) *The activation map of gScoreCAM $M^c_{gScoreCAM}$ (for class c) is then given by:*

$$M^c_{gScoreCAM} = \sum_i w_i^c A_i^c \tag{3.2}$$

where

$$w^c = softmax(s^c) \tag{3.3}$$

and the scores are calculated by

$$s_i^c = \begin{cases} F(I \cdot M_i^{mask})^c, & \textit{if } i \in C_{topk}. \\ 0, & \textit{otherwise}. \end{cases} \tag{3.4}$$

where the top k channels are determined by the gradient ranking;

$$C_{topk} = topIndex(g^c, k) \tag{3.5}$$

Where g^c is the mean value of the gradient of each channel for class c, s^c is the confidence score of the model output (for class c). $F(\cdot)$ denotes the CNN output, A_i is the feature map i^{th} (output of the targeted convolutional layer), $i \in C$, C is the number of channels in the targeted convolutional layer.

3.3 Experiment setups

3.3.1 Datasets & Localization evaluation metrics

We conduct our main experiments on three datasets: ImageNet-v2 [84], PartImageNet [39], and COCO [56]. We use the 2017 COCO dataset and all experiments are tested in the validation set except the experiment in Section 6.2.3. We also use some randomly sampled images of the training set to search for the binarization threshold. For the experiment in Section 6.2.5 we use the combination of labels in the LVIS and COCO training set.

BoxAcc metric For PartImageNet and COCO, we follow Gupta et al. [37] which measures the intersection over union (IoU) between the predicted box and the ground truth box under a certain threshold. The box accuracy is defined as:

$$BoxAcc(\delta) := \frac{1}{N}\sum_{n} 1_{IoU(X_n,B_n)\geq\delta} \tag{3.6}$$

Where X is the predicted box, B is the ground truth box, and δ is the IoU threshold, we use $\delta = 0.5$ in our BoxAcc.

MaxBoxAccV2 metric For ImageNet-v2, we adopt the recommended evaluation metric $MaxBoxAccV2$ proposed by Junsuk et al. [19]:

$$MaxBoxAccV2(\delta) := \frac{1}{3}\sum_{\delta} max_{\tau} BoxAcc(\tau, \delta) \tag{3.7}$$

where $\delta \in \{0.3, 0.5, 0.7\}$, τ is the binary threshold for the resulting heatmap and $\tau \in [0 : 0.05 : 0.95]$. We use the default hyperparameters from the authors.

We also directly measure the IoU (Section 6.2.5) as well as the area under the curve (AUC) of IoU under different binary thresholds (Section 6.2.3) for the predicted bounding box to better understand the difference between multiple types of visualization methods.

Inferring a bounding box from a heatmap

For the evaluation of ImageNet-v2, we follow the traditional procedure as described in [84, 27, 2] that uses the tightest bounding box at binary threshold $\tau \in [0 : 0.05 : 0.95]$ and consider the box that has the highest IoU with ground truth box as the predicted box. For the evaluation of COCO and PartImageNet, we use an alternative approach that is similar to the approach used in Chefer et al. [75]. We employ Otsu's method to binarize the heatmap instead of an exhaustive search for the threshold based on the ground truth (see more detailed description of the Otsu-based evaluation in Section 6.2.8).

We conduct an ablation study on the COCO validation set in Section 6.2.8 and find that the use of searched optimal binary threshold does not improve significantly and the overall trend between the different methods is the same. Therefore, we use Otsu's method for both experiments of PartImageNet and COCO because of its flexibility.

3.3.2 CLIP networks

Model & Methods All experiments are carried out on the basis of the CLIP [80] model. We use the pre-trained model **RN50x16** for all CNN-based methods (except for experiment in Section 6.2.6) as it gives the best performance among the available convolutional-based variances. For comparison, we also apply the CNN-based saliency method to CLIP **ViT-B/32** by reshaping the embedding of the target layer.

Target layers for visualization We use these layers to derive heatmaps and implement visualization methods.

- RN50x16 and RN50x4: We use relu3 of the last BottleNeck in layer4, which is the last layer of the image encoder in CLIP. RN50x16 has 3072 channels with spatial dimension 12×12. RN50x4 has 2560 channels with spatial dimension 9×9.

- ViT-B/32: We use the second-last ResidualAttentionBlock in VisionTransformer. The output dimension is: $50 \times 1 \times 768$, which after excluding the [CLS] vector, is reshaped into $7 \times 7 \times 768$ for CAM-based visualizations.

We choose the second-last layer in ViT-B/32 because the gradients in the last layer are zero except for the [CLS] vector, and, only the [CLS] in the last layer embedding is used for the final prediction. Note that to visualize the embedding in ViT, we discard the [CLS] vector and reshape the embedding from $50 \times 1 \times 768$ to $768 \times 7 \times 7$.

The implementation of our attribution methods for CNNs is from PyTorchCAM [33]. The CLIP models we used are from OpenAI [69]. Our HilaCAM implementation is from the code released by [12].

CLIP models We use CLIP RN50x16, RN50x4, and ViT-B/32 for our experiments. We list some key hyperparameters of the models in Table 3.1. More hyperparameters can be found in Table 19 of Radford et al. [80].

Table 3.1: Some key hyperparameters of the CLIP models [80] used in our experiments.

	Embedding	Input	ResNet-50		Vision Transformer (ViT)		
	dimension	resolution	blocks	width	layers	width	heads
RN50x4	640	288	(4,6,10,6)	2560	N/A		
RN50x16	768	384	(6,8,18,8)	3072			
ViT-B/32	512	224	N/A		12	768	12

Prompts of CLIP For ImageNet-v2 and COCO, we directly use "`{class name}`" (without quotation marks) as the prompt. For PartImageNet, we use “`{class name} {part name}`" as the prompt.

Table 3.2: CLIP zero-shot object localization results with different CAM variances. ScoreCAM and gScoreCAM are similar in overall, but gScoreCAM is much faster in run-time. By default, the results are for CLIP RN50x16 unless otherwise noted (ViT-B/32). Note that gScoreCAM on RN-50x16 is substantially better HilaCAM [14], a state-of-the-art method for CLIP ViT-B/32. In contrast, gScoreCAM on ViT-B/32 performs on par with HilaCAM on ViT-B/32.

	ImageNet-v2 [84] (MaxBoxAcc)	COCO [56] (BoxAcc)	PartImageNet [39] (BoxAcc)	Run time (s)	Number of Forward passes	Number of Backward passes
GradCAM [91]	38.90	11.59	10.91	0.21	1	1
xGradCAM [28]	24.24	5.60	2.93	0.24	1	1
GradCAM++ [11]	44.15	9.68	6.57	0.39	1	1
LayerCAM [46]	43.70	9.19	12.42	0.82	1	1
GroupCAM [115]	50.85	13.06	6.16	1.99	96	1
RISE [75]	41.39	7.26	8.69	166.57	8001	0
HilaCAM [13] (ViT-B/32)	47.79	12.82	11.80	0.26	1	1
gScoreCAM (ViT-B/32)	45.26	12.73	10.67	0.84	301	1
ScoreCAM [105]	**57.78**	20.43	15.76	55.75	3073	0
gScoreCAM (ours)	56.61	**20.83**	**16.34**	7.40	301	1

3.4 gScoreCAM is the State-of-the-art CLIP Visualizer

As indicated in Table 3.2, gScoreCAM establishes itself as a state-of-the-art tool in COCO and PartImageNet, achieving the highest localization accuracies among compared methods. Although its performance on ImageNet-v2 is marginally lower than ScoreCAM, gScoreCAM's runtime efficiency significantly enhances its utility, operating 8 times faster.

Our analyses demonstrate that gScoreCAM excels in localization accuracy and computational efficiency across various datasets. It particularly shines in complex scenarios with multiple objects and diverse object sizes, outperforming existing CAM-based methods. Notably, gScoreCAM delivers clearer and more interpretable heatmaps, effectively highlighting finer details and smaller objects—key factors for enhanced model interpretability.

Qualitative evaluations reinforce these findings, showing gScoreCAM's superior performance in delineating pertinent features within images. Detailed discussions and additional analyses are provided in Section 6.2.1, underscoring gScoreCAM's capabilities as a robust tool for visualizing CLIP models across different backbones and datasets.

3.5 Visualizing CLIP's Attention

In Section 3.4, we demonstrate that gScoreCAM significantly enhances our understanding of CLIP's attention mechanisms by providing state-of-the-art visualization capabilities. This section will focus on how gScoreCAM can help us understand CLIP.

3.5.1 Better Understanding of "Typographic Attack"

Our method provides insightful revelations into CLIP's inner workings, particularly in the context of "typographic attacks," where text within an image can misleadingly influence the model's predictions. As illustrated in Figure 3.2, such attacks can lead to notable failures in detection accuracy by exploiting the model's text-image alignment mechanisms.

Through applying gScoreCAM, we uncover that despite the erroneous outcomes often associated with these attacks, CLIP retains the ability to adjust its focus dynamically in response to the textual prompts. This observation is crucial as it demonstrates that the underlying misclassifications are not due to a lack of recognition capacity but rather the softmax layer's prioritization in the presence of competing visual cues. Our visualizations suggest that CLIP can indeed discern and separately attend to multiple relevant objects within the scene, such as apples and iPods, alongside background elements.

This ability highlights the robust potential of foundation models like CLIP for open-vocabulary tasks, suggesting they possess inherent capabilities to handle complex, multi-object scenarios that are typically challenging for conventional image recognition systems. The visualization provided by gScoreCAM thus not only enhances our understanding of typographic attacks but underscores the sophistication of CLIP's multimodal processing.

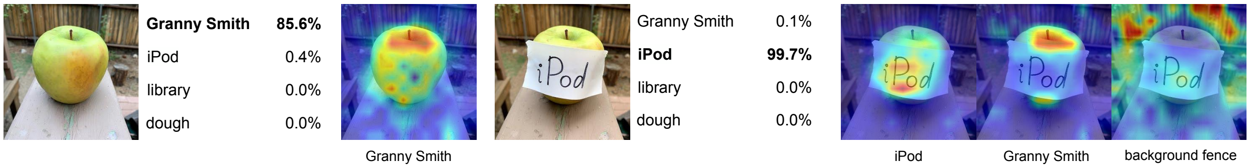


Figure 3.2: While Goh et al. [35] reported that CLIP is easily fooled by *typographic attacks*, our gScoreCAM visualizations reveal interesting insights that CLIP indeed was able to distinguish between apple, iPod and even the background objects. The misclassification was merely because multiple objects are in the scene (i.e., ill-posed, single-label, image classification task).

3.5.2 Inspecting Bias in CLIP

CLIP's training on a large corpus of unfiltered web data aims to enhance its generalization capabilities across various image recognition tasks. However, this approach can inadvertently incorporate and perpetuate societal biases in the training data.

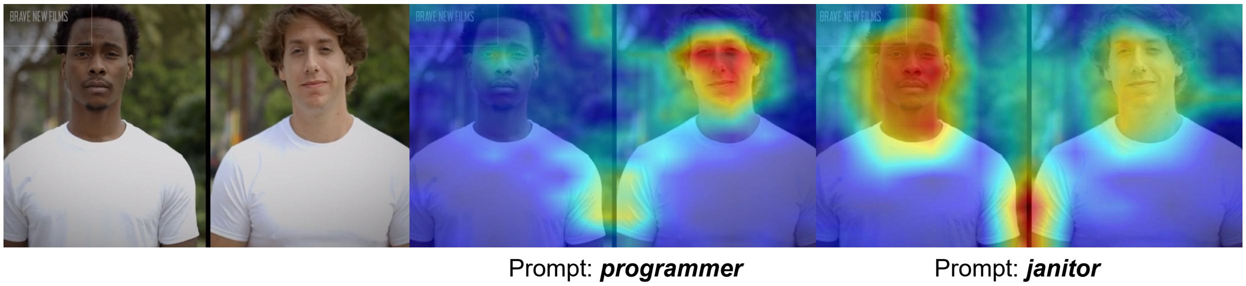


Figure 3.3: Illustration of occupational bias in CLIP: The model tends to associate racial stereotypes with specific professions. Here, CLIP classifies a white man as a "programmer" and a black man as a "janitor", highlighting the need to evaluate training data and model outputs critically.

Our application of gScoreCAM provides a novel lens through which to examine these biases more closely. By visualizing how CLIP focuses its attention when processing images, gScoreCAM reveals patterns that might explain the model's decision-making processes. For instance, the visualizations generated by gScoreCAM show clear differences in attention allocation based on racial features when predicting occupational roles, as seen in Figure 3.3.

This capability underscores the importance of using visualization tools like gScoreCAM to detect and understand biases in AI models and emphasizes the necessity for diverse and inclusive training datasets. By bringing these biases to light, we can better understand the limitations of current models and work towards more equitable AI systems. Our findings stress the need for developers and researchers to incorporate de-biasing techniques and ethical considerations in developing and deploying AI technologies.

3.6 Conclusion

gScoreCAM has proven to be a valuable tool in elucidating the inner workings of CLIP, providing critical insights into its strengths and the inherent biases that arise from its training on extensive web data. While gScoreCAM effectively highlights areas for concern, such as embedded societal biases, it is important to recognize the limitations inherent in any post-hoc explanatory method. These techniques offer insights after making model decisions and do not inherently change the underlying model behavior.

Addressing these deep-seated biases is challenging; it typically requires extensive filtering of training datasets and potentially costly retraining of models. Such measures, while effective, pose significant resource demands and logistical challenges.

The upcoming chapter will explore an innovative, self-explainable framework to mitigate these issues. This framework aims to empower users to understand and rectify biases directly, offering a more dynamic and interactive approach to model transparency and accountability. Through this development, we seek to bridge the gap between post-hoc explanation and proactive intervention, enhancing the ethical deployment of AI systems in diverse real-world applications.

Chapter 4

PEEB: A Part-based Image Classifiers with an Explainable and Editable Language Bottleneck

4.1 The Necessity and Benefits of Self-Explainable Frameworks

The rapid advancement and deployment of artificial intelligence (AI) systems, especially in computer vision, have achieved remarkable feats, often surpassing human capabilities in tasks like fine-grained image classification. However, the opacity inherent in these "black-box" systems poses significant barriers to their wider acceptance, particularly in scenarios where trust and understanding are paramount. Our proposed framework PEEB (**P**art-based image classifier with an **E**xplainable and **E**ditable via a natural-language **B**ottleneck) emerges as a vital solution to these challenges, underscoring the necessity of self-explainable systems in fostering trust and enhancing user interaction.

Enhancing Transparency, Trust, and Model Understanding The PEEB framework increases AI transparency by mapping visual parts to natural-language descriptors, allowing users to understand and question AI decisions. This capability is crucial in fields requiring precise and justifiable decisions, such as wildlife conservation and academic research [36, 86, 104]. Integrating this human-like understanding into prediction models provides comprehensible explanations and uses these insights to refine model robustness and accuracy [22, 15], aligning AI operations more closely with user expectations and real-world needs.

Facilitating User Interaction and Editability PEEB's design enables users to modify its descriptors directly, allowing for the rapid adaptation of the model to new classes or the correction of errors. This flexibility is particularly beneficial in managing rare or novel species not included in original training datasets, thus bypassing the need for extensive re-training [108, 118].

Addressing Data and Diversity Limitations Utilizing a comprehensive and diverse dataset, PEEB demonstrates how integrating textual descriptions with visual part detection can overcome typical dataset size and species diversity constraints. Its adaptability across various domains, illustrated through its application to bird and dog datasets, underscores its broad utility and scalability [70, 104].

4.2 Datasets

To facilitate the development of PEEB, we established two specialized datasets: Bird-11K and Dog-140. These datasets are crucial for testing the framework's capability to generalize across different domains.

4.2.1 BirdSoup Dataset

The Bird-11K dataset is an extensive collection of bird images aggregated from seven distinct sources, augmented by approximately 55,000 additional images (spanning 10,534 classes) sourced from the Macaulay Library at Cornell. In total, Bird-11K contains 440,934 images covering 11,183 classes, making it arguably the first bird dataset to nearly encompass all bird species known globally. This breadth supports Bird-11K's role as a proof of concept, demonstrating that when our proposed framework is trained on sufficiently large and diverse data, it can effectively generalize to previously unseen classes. The methodology for assembling Bird-11K is not included in this paper; however, we provide a script on Github for those interested in reconstructing the dataset.

4.2.2 DogSoup Dataset

Contrasting with the expansive nature of Bird-11K, Dog-140 is a smaller dataset specifically designed to test the generalization capabilities of PEEB within a different, more restricted domain. This dataset is composed of dog images derived from the ImageNet-21K dataset. Although smaller in scale, Dog-140 is crucial for evaluating the adaptability of PEEB to various animal categories.

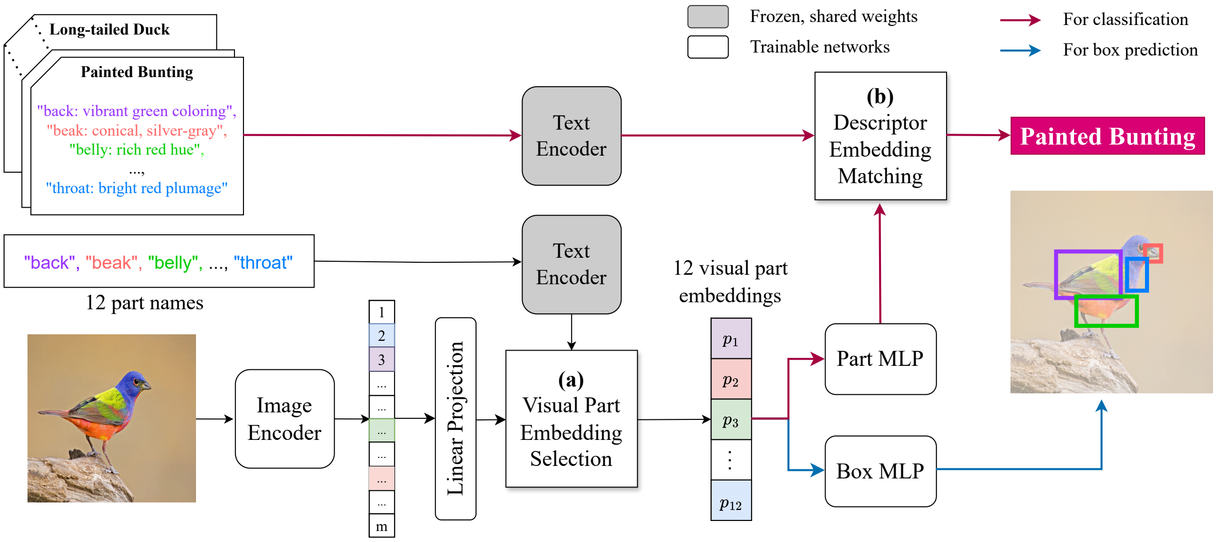


Figure 4.1: During inference, 12 visual part embeddings with the highest cosine similarity with encoded part names are selected **(a)**. These visual part embeddings are then mapped (⟶) to bounding boxes via Box MLP. Simultaneously, the same embeddings are forwarded to the Part MLP and its outputs are then matched **(b)** with textual part descriptors to make classification predictions (⟶). Figure 6.25 shows a more detailed view of the same process.

Details on the construction of both datasets are provided in Section 6.3.4, where we outline the selection criteria, source integration, and preprocessing techniques used to ensure data quality and relevance for training our model.

4.3 Method: PEEB Architecture and Training Strategy

4.3.1 Backbone: OWL-ViT object-part detector

OWL-ViT is an open-vocabulary detector that detects objects and parts in an image given a text prompt, even if the model is not explicitly finetuned to detect those concepts. OWL-ViT consists of four networks (Figure 4.1): (1) a standard ViT-based image encoder, (2) an architecturally identical text encoder, (3) a bounding-box regression head called Box MLP, and (4) and a Linear Projection. Box MLP is a three-layer Multilayer Perceptron (MLP) with a GELU activation [42] after each of the first two layers. The Linear Projection projects the visual embeddings to the same dimensional space with text embeddings (see Fig. 1 in [61]).

4.3.2 PEEB classifier

Architecture PEEB (Figure 4.1) has five networks: an image encoder, a text encoder, a Linear Projection, a Part MLP, and a Box MLP. We introduce **Part MLP** to map the visual part embeddings to the same space with text embeddings for computing dot products (logits) for classification ($\longrightarrow$ in Figure 4.1). This design allows PEEB to easily extend the number of classes without any re-training. Except for Part MLP, all components are adopted from the OWL-ViT framework. Details of all components are in Section 6.3.1.

Inference Given an input image, we first use the 12 generic part names to select the visual part embeddings based on cosine similarity. These selected visual part embeddings are then simultaneously fed into both Part MLP and Box MLP.

Box MLP predicts the bounding box from each part embedding. We compute a dot product to measure the similarity between each embedding output from Part MLP and a corresponding part-descriptor embedding. For classification, a class logit is the sum of the 12 dot products, which essentially computes the distance between the 12 parts in the image and the 12 text descriptors of each class.

4.3.3 Training strategy

Trainable networks In preliminary experiments, we find training only Part MLP (while keeping all other networks frozen) to result in poor accuracy. Therefore, we train Part MLP from scratch and also finetune the image encoder, Linear Projection, and Box MLP. All OWL-ViT components are finetuned from their original weights. In contrast, our proposed Part MLP starts from random weights. Our training has two phases:

1. A two-stage **pre-training** on the large-scale Bird-11K dataset.
2. **Finetuning** on downstream tasks.

More hyperparameter details are in Section 6.3.1.

Objectives We aim to train PEEB to classify images well while maintaining the ability to detect object parts. This translates into **three training objectives**: **(1)** Train the Part MLP contrastively using a symmetric cross-entropy (SCE) loss [80] to maximize the similarity between region-text pairs while minimizing the similarity for negative pairs; **(2)** Train the Linear Projection using a SCE loss to mimic OWL-ViT's behaviors (i.e. the similarity matrix) for part selection; and **(3)** Train Box MLP to predict bounding boxes with DETR losses [116] i.e. a linear combination of ℓ_1 corner-to-corner distance loss and GIoU loss [85]. Loss functions are described in Section 6.3.1.

A challenge When jointly minimizing all three losses above is that PEEB's validation loss improves significantly slowly, perhaps because of some tension between the two SCE losses and the DETR detection loss.

To overcome this challenge, we split the pre-training phase into two stages: (1) first, train the image encoder and Part MLP for classification using the SCE loss; then (2) train the Linear Projection and Box MLP using the 2nd and 3rd loss so they can adapt their weights to the updated image encoder. We always keep the text encoder frozen since we want to preserve its generalizability to the descriptors of unseen objects.

Pre-training on Bird-11K dataset

Stage 1: Contrastive learning The image encoder and Part MLP are jointly trained using a SCE loss, which allows PEEB to learn to map the visual parts to corresponding text descriptors.

In this stage, we use a pre-trained OWL-ViT$_{\text{large}}$ to *select* 12 part embeddings per input image (i.e., teacher forcing) to ensure the *selection* of part embeddings is meaningful and consistent while the embeddings themselves are updating (see Figure 6.26).

Stage 2: Learning to detect from a teacher After the image encoder is modified in Stage 1, we then train the Linear Projection and Box MLP jointly. We use the OWL-ViT$_{\text{large}}$ as the teacher to train both components. Using SCE loss, we train the Linear Projection such that the similarity matrix between the part-names and visual parts matches those of the teacher (Figure 6.27, 1a–c, 2a–c).

Given the absence of human-annotated boxes for object parts, we train Box MLP to predict the same boxes as the predictions by OWL-ViT$_{\text{large}}$ using DETR losses (Figure 6.27, 2d). In this Stage 2, the image encoder is frozen while Part MLP is not involved. **After 2-stage training**, PEEB can perform zero-shot classification while providing the correspondences between visual parts and descriptors as faithful explanations.

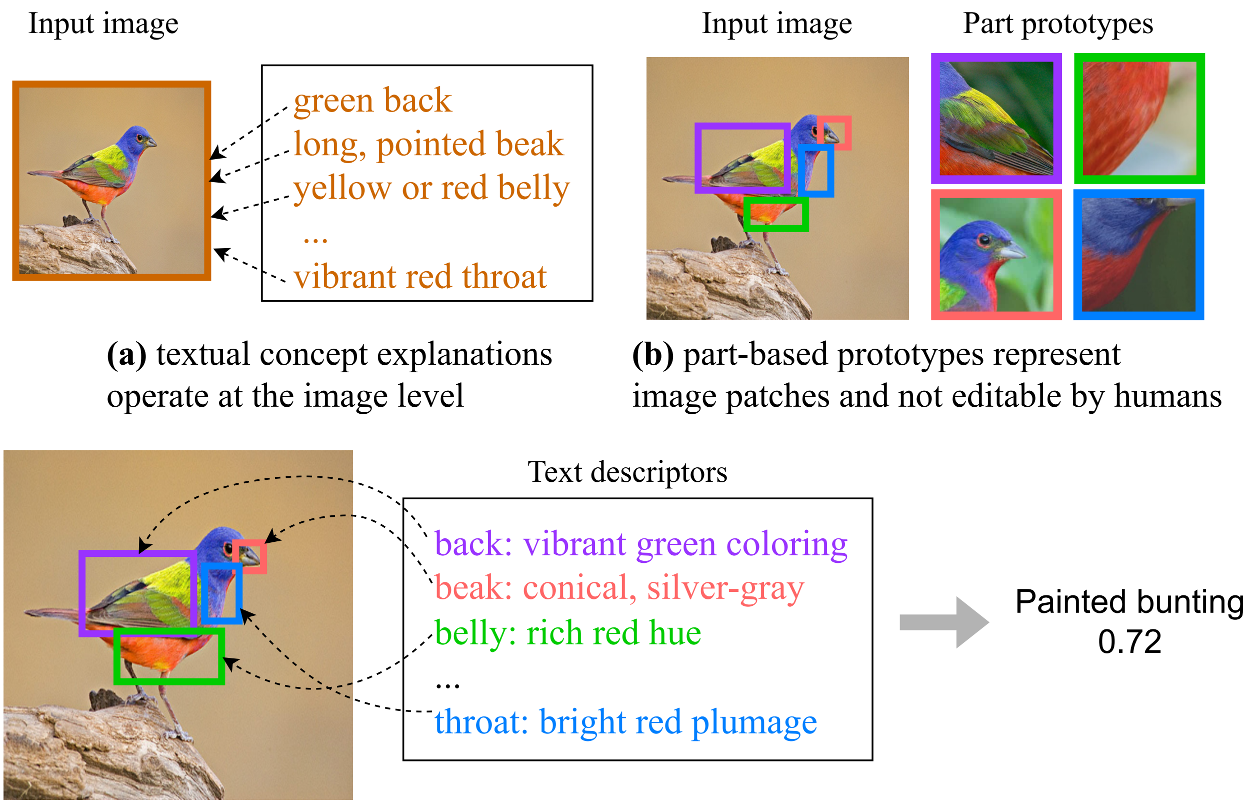
Finetuning on classification tasks

We further finetune the pre-trained PEEB on downstream tasks, e.g., CUB, NABirds, and iNaturalist, to further improve its accuracy. In this phase, to adapt to a downstream task, all components except the text encoder are trained jointly, and the loss for Part MLP is changed from SCE (contrastive) to CE (classification) while other losses are kept intact.

4.4 State of the Art Explainable Bird Classifier

Explainability and Transparency Many bird classifiers, including prominent models cited in the literature [15, 22], claim explainability by comparing the input image with a set of learned part prototypes (Figure 4.2b) or through natural-language concepts (Figure 4.2a). However, these prototypes are essentially feature vectors that users cannot modify. Moreover, when using textual concepts, the comparison often spans the entire image, leaving it unclear which specific image details correspond to a given descriptor [60, 111]. Our PEEB model (Figure 4.2c) innovates by using direct text input and providing detailed text-to-part mappings and scores for each mapping. This approach enhances transparency by showing the classification and rationale behind it, making it superior in clarity to other methods.

State-of-the-Art Performance in Bird Classification Our PEEB model, fine-tuned on the CUB dataset, achieves unprecedented results among explainable classifiers. Initially, when trained with the OWL-ViT$_{\text{base32}}$ backbone, PEEB records a respectable 77.80% accuracy. Pre-training on the comprehensive Bird-11K dataset significantly boosts this performance, reaching accuracies of 86.73% and 88.80% with two different backbones, as detailed in Table 4.1. PEEB not only surpasses



(a) textual concept explanations operate at the image level

(b) part-based prototypes represent image patches and not editable by humans

(c) PEEB explanations pair up each detected object part with a textual descriptor

Figure 4.2: Existing explanations are either (a) textual but at the image level; or (b) part-level but not textual. Combining the best of both worlds, PEEB (c) first matches each detected object part to a text descriptor, then uses the part-level matching scores to classify the image.

the Deformable ProtoPNet, which achieves an 86.4% accuracy, and ProtoTree, with 87.20%, but also establishes a new standard for the field, underscoring PEEB's role as a leading solution in accuracy and explainability in bird classification. The detailed hyperparameters of the fine-tuned model are documented in Table 6.8,

4.5 Editable Part-based Design

PEEB directly utilizes the input prompt for classification. Therefore, the user can easily edit the classifier by changing the definition of a class as shown in Figure 4.3. And unlike CLIP, which responds in trivial differences for different inputs, PEEB correct leverages the input information for better classification as shown in Figure 4.4 (See Section 6.3.7 for more examples). We also find that

Table 4.1: PEEB is a state-of-the-art model (here, top-1 accuracy on CUB-200) w.r.t. explainable classifiers in supervised learning. * *Five ensembled models.*

Methods	Model size	Backbone	Accuracy
Base (ViT) [2021]	22M	DeiT-S [2021]	84.28
– Concept bottleneck classifiers			
Concept Bottleneck Models [2020]	11M	ResNet-18 [2016]	80.10
CPM [2023]	155M	ViT-B/16 [2021]	72.00
CDM [2023]	155M	ViT-B/16	74.31
LaBo [2023]	427M	ViT-L/14	81.90
– Part-based classifiers			
ProtoPNet [2019]	22M	DeiT-S	84.04
ProtoTree [2021]	92M*	ResNet-50 [2016]	87.20*
TesNet [2021]	79M	Dense121 [2017]	84.80
Deformable ProtoPNet [2022]	23M	ResNet-50	86.40
ProtoPFormer [2022]	22M	DeiT-S	84.85
PEEB (**ours**) – w/o pre-training	155M	OWL-ViT$_{\text{base32}}$	**77.80**
PEEB (**ours**)	155M	OWL-ViT$_{\text{base32}}$	**86.73**
PEEB$_{B16}$ (**ours**)	155M	OWL-ViT$_{\text{base16}}$	**88.80**

a more precise input will result in better classifications (Section 6.3.6). However, due to the fact that there is a lot of noise in our training data, the current version does not guarantee to provide correct classification even if the class definition is correct.

4.6 Generalize to Unseen Classes

The ability to generalize to unseen classes is a crucial aspect of vision-language models [45, 118, 74, 18, 51, 83]. In our evaluation, we utilize the ZSL (Zero-Shot-Learning) split as defined by Akata et al. on the CUB dataset, along with the Super-Category-Similar/Exclusive (SCS/SCE) splits proposed by Elhoseiny et al. on both CUB and NABirds. These splits introduce two levels of difficulty in the ZSL test: SCS (easy) and SCE (hard), intentionally designed based on species hierarchies to assess different generalization capabilities.

Adhering to the traditional ZSL setting, we exclude all classes from CUB and NABirds during pre-training and finetune the model using the splits designed by Akata et al. and Elhoseiny et al..

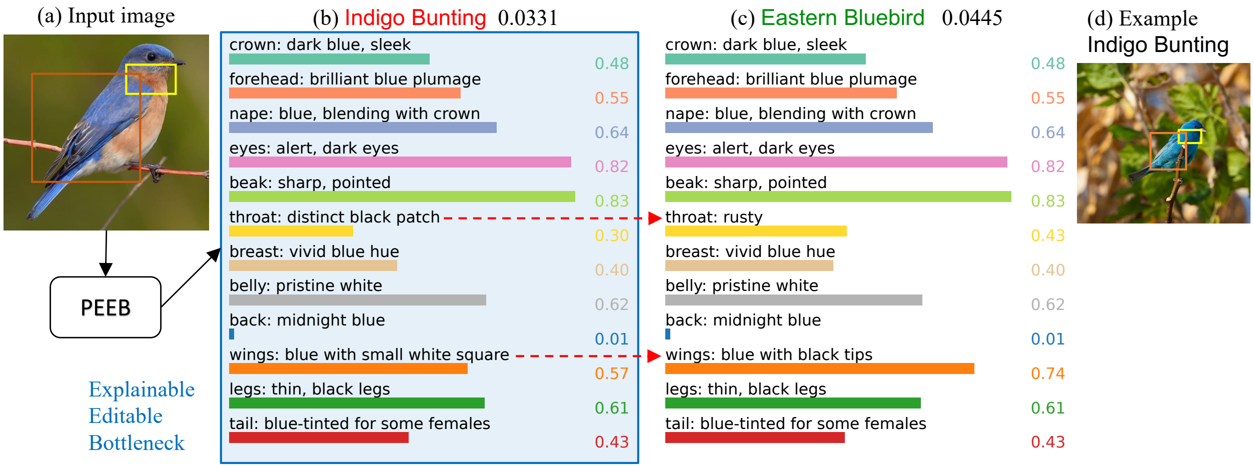

(a) Input image
PEEB
Explainable
Editable
Bottleneck
(b) Indigo Bunting 0.0331
crown: dark blue, sleek 0.48
forehead: brilliant blue plumage 0.55
nape: blue, blending with crown 0.64
eyes: alert, dark eyes 0.82
beak: sharp, pointed 0.83
throat: distinct black patch 0.30
breast: vivid blue hue 0.40
belly: pristine white 0.62
back: midnight blue 0.01
wings: blue with small white square 0.57
legs: thin, black legs 0.61
tail: blue-tinted for some females 0.43
(c) Eastern Bluebird 0.0445
crown: dark blue, sleek 0.48
forehead: brilliant blue plumage 0.55
nape: blue, blending with crown 0.64
eyes: alert, dark eyes 0.82
beak: sharp, pointed 0.83
throat: rusty 0.43
breast: vivid blue hue 0.40
belly: pristine white 0.62
back: midnight blue 0.01
wings: blue with black tips 0.74
legs: thin, black legs 0.61
tail: blue-tinted for some females 0.43
(d) Example
Indigo Bunting

Figure 4.3: Given an input image (a) from an unseen class of EasternBluebird, PEEB misclassifies it into IndigoBunting (b), a visually similar blue bird in CUB-200 (d). To add a new class for EasternBluebird to the 200-class list that PEEB considers when classifying, we clone the 12 textual descriptors of IndigoBunting (b) and **edit** (- - ▸) the descriptor of throat and wings (c) to reflect their identification features described on AllAboutBirds.org *("Male Eastern Bluebirds are vivid, deep blue above and rusty or brick-red on the throat and breast")*. After the edit, PEEB correctly predicts the input image into EasternBluebird (0.0445) out of 201 classes (c).

We allocate approximately 10% of the training set for validation and select checkpoints based on the lowest validation loss.

PEEB significantly outperforms all baselines across three test splits (from CUB and NABirds) by a margin of (+4 to +10 points) in terms of harmonic mean. This performance indicates a robust generalization of PEEB not only to seen classes, where it achieves an 80.78 v.s. 65.80 lead, but also to unseen classes, as detailed in Table 4.2. In the easier SCS tests, where the presence of classes is similar to those in the training set, models that perform well on the training data typically show high accuracy. However, the more challenging SCE splits, which feature test classes from categories distinct from those in the training set, provide a stringent measure of a model's generalization ability. Here, PEEB excels over all baselines by (+5 to +15 points) in accuracy for the SCE split and by +2.64 points compared to CLORE$_{CLIP}$.

Moreover, in a further evaluation without any CUB and NABirds classes in the pre-training phase, PEEB surpasses the performance of both CLIP and M&V methods on the CUB$_{sci}$ and NABirds$_{sci}$ scenarios. Specifically, PEEB outperforms these baselines by (+10 to +12 points) on

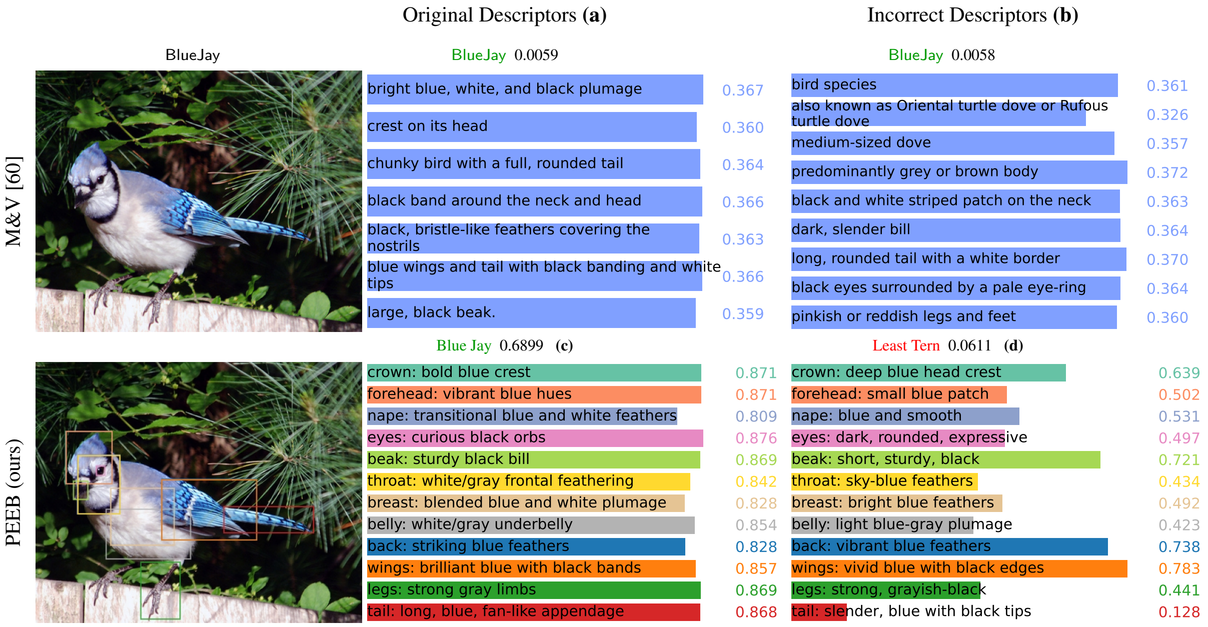


Figure 4.4: With original descriptors, M&V [60] correctly classifies the input image into BlueJay **(a)**. Yet, interestingly, when randomly swapping the descriptors of this class with those of other classes **(b)**, M&V's top-1 prediction remains unchanged, suggesting that the class names (hidden) in the prompt have the most influence over the prediction (not the expressive descriptors). In contrast, PEEB changes its top-1 prediction from BlueJay (c) to LeastTern when the descriptors are randomized (d).

CUB and (+1 to +3 points) on NABirds, as shown in Section 6.3.5. These findings solidly affirm the superior generalization capability of our method.

Table 4.2: PEEB consistently outperforms other vision-language methods under Harmonic mean and especially in the hard split (SCE) by (+5 to +15) points, highlighting its generalization capability on ZSL.

Methods	CUB			NABirds		
	Seen	Unseen	Harmonic	Seen	Unseen	Harmonic
Data split by Akata et al. [3]						
CLORE$_{CLIP}$ [2023]	65.80	39.10	49.05	n/a		
PEEB (**ours**)	**80.78**	**41.74**	**55.04**			
SCS/SCE splits by Elhoseiny et al. [24]						
	SCS (Easy)	SCE (Hard)	Harmonic	SCS (Easy)	SCE (Hard)	Harmonic
S^{2}GA-DET [2018]	42.90	10.90	17.38	39.40	9.70	15.56
GRZSL [2018]	44.08	14.46	21.77	36.36	9.04	14.48
ZEST [2020]	**48.57**	15.26	23.22	38.51	10.23	16.17
CANZSL [2020]	45.80	14.30	21.12	38.10	8.90	14.43
DGRZSL [2021]	45.48	14.29	21.75	37.62	8.91	14.41
DPZSL [2023]	45.40	15.50	23.11	**40.80**	8.20	13.66
PEEB (**ours**)	44.66	**20.31**	**27.92**	28.26	**24.34**	**26.15**

4.7 Conclusion

PEEB stands out as a transparent and editable classifier to users by grounding the text descriptors to the visual parts in the image (Figure 4.4-bottom). This transparent decision-making process plays an important role in user understanding. For instance, in Figure 4.4 (upper right), the challenge of understanding why the model predicts accurately persists, particularly when we already know that the descriptors are incorrect. In contrast, PEEB's explanations not only make errors like the mismatch between throat and wings more apparent but also enable users to adjust descriptions, thereby improving model accuracy without the need for re-training (Figure 4.3).

We introduce PEEB, a pioneering part-based, explainable, and editable image classifier that leverages textual descriptors for bird parts. By grounding natural language descriptors with visual features, PEEB brings transparency to its decision-making. Besides, PEEB achieves superior performance in both GZSL and ZSL settings compared to existing state-of-the-art explainable models.

Moreover, we contribute to the broader research community by developing the Bird-11K dataset, which encompasses a diverse range of bird species and presents a valuable resource for further explorations in fine-grained classification and beyond.

4.8 Limitations

Text encoder may not fully comprehend the bird descriptors Our text encoder, pre-trained on a broad image-text dataset, may not fully capture the intricate details specific to birds. Furthermore, CLIP text encoders trained by contrastive learning are known to suffer from the *binding* problem and do not understand some logical operators such as “and”, “or”, or negation. PEEB accuracy depends directly on the quality of the text encoder.

Dependency on image encoder for part visibility The image encoder’s role in determining the visibility of bird parts in an image poses another limitation. Our model operates under the assumption that 12 parts are always visible in a bird image, requiring it to score these parts even when they might not be visually present. In an ideal scenario, the model should learn to assign the absence part a low score. This approach, which lacks direct supervision, relies heavily on unsupervised learning derived from class labels. Consequently, the limited dataset size of approximately 290K training images in Bird-11K may not sufficiently support robust unsupervised learning.

Hallucinations in GPT-4 descriptors The accuracy of our model is directly impacted by the quality of GPT-4 descriptors. Our empirical analysis across 20 bird classes revealed that, on average, 45% of these descriptors do not accurately reflect the birds’ features (Section 6.3.6). However, we observed that revising certain descriptors in the CUB dataset led to a significant improvement of

+10 points in classification accuracy for those classes (Section 6.3.6). This primitive observation suggests that PEEB can be further improved if trained with human-labeled descriptors.

Chapter 5

Reference

[1] E. Aflalo, M. Du, S.-Y. Tseng, Y. Liu, C. Wu, N. Duan, and V. Lal. Vl-interpret: An interactive visualization tool for interpreting vision-language transformers. *arXiv preprint arXiv:2203.17247*, 2022.

[2] C. Agarwal and A. Nguyen. Explaining image classifiers by removing input features using generative models. In *Proceedings of the Asian Conference on Computer Vision*, 2020.

[3] Z. Akata, F. Perronnin, Z. Harchaoui, and C. Schmid. Label-embedding for image classification. *IEEE transactions on pattern analysis and machine intelligence*, 38(7):1425–1438, 2015.

[4] N. Bansal, C. Agarwal, and A. Nguyen. Sam: The sensitivity of attribution methods to hyperparameters. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2020.

[5] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6541–6549, 2017.

[6] T. Berg, J. Liu, S. Woo Lee, M. L. Alexander, D. W. Jacobs, and P. N. Belhumeur. Birdsnap: Large-scale fine-grained visual categorization of birds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2011–2018, 2014.

[7] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[8] W. Brendel and M. Bethge. Approximating CNNs with bag-of-local-features models works surprisingly well on imagenet. In *International Conference on Learning Representations*, 2019.

[9] H. Caesar, J. Uijlings, and V. Ferrari. Coco-stuff: Thing and stuff classes in context. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1209–1218, 2018.

[10] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020.

[11] A. Chattopadhay, A. Sarkar, P. Howlader, and V. N. Balasubramanian. Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In *2018 IEEE winter conference on applications of computer vision (WACV)*, pages 839–847. IEEE, 2018.

[12] H. Chefer, S. Gur, and L. Wolf. Generic attention-model explainability for interpreting bi-modal and encoder-decoder transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 397–406, October 2021.

[13] H. Chefer, S. Gur, and L. Wolf. Generic attention-model explainability for interpreting bi-modal and encoder-decoder transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 397–406, 2021.

[14] H. Chefer, S. Gur, and L. Wolf. Transformer interpretability beyond attention visualization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 782–791, 2021.

[15] C. Chen, O. Li, D. Tao, A. Barnett, C. Rudin, and J. K. Su. This looks like that: deep learning for interpretable image recognition. *Advances in neural information processing systems*, 32, 2019.

[16] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[17] P. Chen, Q. Li, S. Biaz, T. Bui, and A. Nguyen. gscorecam: What objects is clip looking at? In *Proceedings of the Asian Conference on Computer Vision*, pages 1959–1975, 2022.

[18] Z. Chen, J. Li, Y. Luo, Z. Huang, and Y. Yang. Canzsl: Cycle-consistent adversarial networks for zero-shot learning from natural language. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 874–883, 2020.

[19] J. Choe, S. J. Oh, S. Lee, S. Chun, Z. Akata, and H. Shim. Evaluating weakly supervised object localization methods right. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3133–3142, 2020.

[20] R. M. Cichy, A. Khosla, D. Pantazis, A. Torralba, and A. Oliva. Comparison of deep neural networks to spatio-temporal cortical dynamics of human visual object recognition reveals hierarchical correspondence. *Scientific reports*, 6(1):1–13, 2016.

[21] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[22] J. Donnelly, A. J. Barnett, and C. Chen. Deformable protopnet: An interpretable image classifier using deformable prototypes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10265–10275, 2022.

[23] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

[24] M. Elhoseiny, Y. Zhu, H. Zhang, and A. Elgammal. Link the head to the" beak": Zero shot learning from noisy text description at part precision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5640–5649, 2017.

[25] L. Engstrom, A. Ilyas, S. Santurkar, and D. Tsipras. Robustness (python library), 2019. URL https://github.com/MadryLab/robustness.

[26] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, B. Tran, and A. Madry. Adversarial robustness as a prior for learned representations, 2020. URL https://openreview.net/forum?id=rygvFyrKwH.

[27] R. C. Fong and A. Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE international conference on computer vision*, pages 3429–3437, 2017.

[28] R. Fu, Q. Hu, X. Dong, Y. Guo, Y. Gao, and B. Li. Axiom-based grad-cam: Towards accurate visualization and explanation of cnns. *arXiv preprint arXiv:2008.02312*, 2020.

[29] Fédération Cynologique Internationale (FCI). Nomenclature of the breeds recognised by the fci, 2023. URL https://www.fci.be/en/Nomenclature/. Accessed: 2014-02-25.

[30] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.

[31] R. Geirhos, C. R. Temme, J. Rauber, H. H. Schütt, M. Bethge, and F. A. Wichmann. Generalisation in humans and deep neural networks. In *Advances in Neural Information Processing Systems*, pages 7538–7550, 2018.

[32] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel. Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=Bygh9j09KX`.

[33] J. Gildenblat and contributors. Pytorch library for cam methods. `https://github.com/jacobgil/pytorch-grad-cam`, 2021.

[34] J. Gilmer, N. Ford, N. Carlini, and E. Cubuk. Adversarial examples are a natural consequence of test error in noise. In *International Conference on Machine Learning*, pages 2280–2289, 2019.

[35] G. Goh, N. Cammarata, C. Voss, S. Carter, M. Petrov, L. Schubert, A. Radford, and C. Olah. Multimodal neurons in artificial neural networks. *Distill*, 6(3):e30, 2021.

[36] D. Gunning, E. Vorm, J. Y. Wang, and M. Turek. Darpa's explainable ai (xai) program: A retrospective. *Applied AI Letters*, 2(4):e61, 2021. doi: https://doi.org/10.1002/ail2.61. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/ail2.61`.

[37] T. Gupta, A. Vahdat, G. Chechik, X. Yang, J. Kautz, and D. Hoiem. Contrastive learning for weakly supervised phrase grounding. In *European Conference on Computer Vision*, pages 752–768. Springer, 2020.

[38] C. Han, H. Pei, X. Du, and H. Ji. Zero-shot classification by logical reasoning on natural language explanations. In A. Rogers, J. Boyd-Graber, and N. Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8967–8981, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.571. URL `https://aclanthology.org/2023.findings-acl.571`.

[39] J. He, S. Yang, S. Yang, A. Kortylewski, X. Yuan, J.-N. Chen, S. Liu, C. Yang, Q. Yu, and A. Yuille. Partimagenet: A large, high-quality dataset of parts. In *European Conference on Computer Vision*, pages 128–145. Springer, 2022.

[40] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[41] D. Hendrycks and T. Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HJz6tiCqYm`.

[42] D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[43] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[44] ImageMagick. Imagemagick, 2020. URL `https://imagemagick.org`.

[45] Z. Ji, Y. Fu, J. Guo, Y. Pang, Z. M. Zhang, et al. Stacked semantics-guided attention model for fine-grained zero-shot learning. *Advances in neural information processing systems*, 31, 2018.

[46] P.-T. Jiang, C.-B. Zhang, Q. Hou, M.-M. Cheng, and Y. Wei. Layercam: Exploring hierarchical class activation maps for localization. *IEEE Transactions on Image Processing*, 30: 5875–5888, 2021.

[47] A. Khosla, N. Jayadevaprakash, B. Yao, and F.-F. Li. Novel dataset for fine-grained image categorization: Stanford dogs. In *Proc. CVPR workshop on fine-grained visual categorization (FGVC)*, volume 2. Citeseer, 2011.

[48] G. Kim and J. C. Ye. Diffusionclip: Text-guided image manipulation using diffusion models. *arXiv preprint arXiv:2110.02711*, 2021.

[49] W. Kim, B. Son, and I. Kim. Vilt: Vision-and-language transformer without convolution or region supervision. In *International Conference on Machine Learning*, pages 5583–5594. PMLR, 2021.

[50] P. W. Koh, T. Nguyen, Y. S. Tang, S. Mussmann, E. Pierson, B. Kim, and P. Liang. Concept bottleneck models. In *International conference on machine learning*, pages 5338–5348. PMLR, 2020.

[51] S. Kousha and M. A. Brubaker. Zero-shot learning with class description regularization. *arXiv preprint arXiv:2106.16108*, 2021.

[52] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[53] G. Kwon and J. C. Ye. Clipstyler: Image style transfer with a single text condition. *arXiv preprint arXiv:2112.00374*, 2021.

[54] J. Lei, L. Li, L. Zhou, Z. Gan, T. L. Berg, M. Bansal, and J. Liu. Less is more: Clipbert for video-and-language learning via sparse sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7331–7341, 2021.

[55] Q. Li, L. Mai, M. A. Alcorn, and A. Nguyen. A cost-effective method for improving and re-purposing large, pre-trained gans by fine-tuning their class-embeddings. In *Proceedings of the Asian Conference on Computer Vision*, 2020.

[56] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[57] Z. C. Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

[58] H. Luo, L. Ji, M. Zhong, Y. Chen, W. Lei, N. Duan, and T. Li. Clip4clip: An empirical study of clip for end to end video clip retrieval. *arXiv preprint arXiv:2104.08860*, 2021.

[59] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJzIBfZAb`.

[60] S. Menon and C. Vondrick. Visual classification via description from large language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=jlAjNL8z5cs`.

[61] M. Minderer, A. Gritsenko, A. Stone, M. Neumann, D. Weissenborn, A. Dosovitskiy, A. Mahendran, A. Arnab, M. Dehghani, Z. Shen, X. Wang, X. Zhai, T. Kipf, and N. Houlsby. Simple open-vocabulary object detection with vision transformers. *ECCV*, 2022.

[62] M. Nauta, R. Van Bree, and C. Seifert. Neural prototype trees for interpretable fine-grained image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14933–14943, 2021.

[63] nerdyrodent. nerdyrodent/vqgan-clip: Just playing with getting vqgan+clip running locally, rather than having to use colab. `https://github.com/nerdyrodent/VQGAN-CLIP`, 7 2022. (Accessed on 05/18/2022).

[64] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[65] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In *Advances in neural information processing systems*, pages 3387–3395, 2016.

[66] A. Nguyen, J. Yosinski, and J. Clune. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. In *Visualization for Deep Learning Workshop, ICML conference*, 2016.

[67] A. Nguyen, J. Yosinski, and J. Clune. Understanding neural networks via feature visualization: A survey. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 55–76. Springer, 2019.

[68] T. Oikarinen, S. Das, L. M. Nguyen, and T.-W. Weng. Label-free concept bottleneck models. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=FlCg47MNvBA`.

[69] OpenAI. openai/clip: Contrastive language-image pretraining. `https://github.com/openai/CLIP`, 07 2022. (Accessed on 07/06/2022).

[70] OpenAI. Gpt-4 technical report, 2023.

[71] K. P. Panousis, D. Ienco, and D. Marcos. Hierarchical concept discovery models: A concept pyramid scheme. *arXiv preprint arXiv:2310.02116*, 2023.

[72] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library`
`pdf`.

[73] O. Patashnik, Z. Wu, E. Shechtman, D. Cohen-Or, and D. Lischinski. Styleclip: Text-driven manipulation of stylegan imagery. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2085–2094, 2021.

[74] T. Paz-Argaman, R. Tsarfaty, G. Chechik, and Y. Atzmon. ZEST: Zero-shot learning from text descriptions using textual similarity and visual summarization. In T. Cohn, Y. He, and Y. Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 569–579, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.50. URL `https://aclanthology.org/2020.findings-emnlp.50`.

[75] V. Petsiuk, A. Das, and K. Saenko. Rise: Randomized input sampling for explanation of black-box models. *arXiv preprint arXiv:1806.07421*, 2018.

[76] T. M. Pham, P. Chen, T. Nguyen, S. Yoon, T. Bui, and A. Nguyen. Peeb: Part-based image classifiers with an explainable and editable language bottleneck. *arXiv preprint arXiv:2403.05297*, 2024.

[77] P. J. Phillips, C. A. Hahn, P. C. Fontana, D. A. Broniatowski, and M. A. Przybocki. Four principles of explainable artificial intelligence. *Gaithersburg, Maryland*, 2020.

[78] G. Piosenka. Birds 525 - species image classification. 02 2022. URL `https://www.kaggle.com/datasets/gpiosenka/100-bird-species`.

[79] PyTorch. torchvision.models — pytorch master documentation. `https://pytorch.org/docs/stable/torchvision/models.html`, 2019. (Accessed on 09/21/2019).

[80] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[81] L. Radin, S. Osher, and E. Fatemi. Non-linear total variation noise removal algorithm. *Phys D*, 60:259–268, 1992.

[82] R. Rajalingham, E. B. Issa, P. Bashivan, K. Kar, K. Schmidt, and J. J. DiCarlo. Large-scale, high-resolution comparison of the core visual object recognition behavior of humans, monkeys, and state-of-the-art deep artificial neural networks. *Journal of Neuroscience*, 38 (33):7255–7269, 2018.

[83] Y. Rao, Z. Yang, S. Zeng, Q. Wang, and J. Pu. Dual projective zero-shot learning using text descriptions. *ACM Transactions on Multimedia Computing, Communications and Applications*, 19(1):1–17, 2023.

[84] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar. Do imagenet classifiers generalize to imagenet? In *International Conference on Machine Learning*, pages 5389–5400. PMLR, 2019.

[85] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[86] C. Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence*, 1(5):206–215, 2019.

[87] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268, 1992.

[88] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[89] H. Salman, A. Ilyas, L. Engstrom, A. Kapoor, and A. Madry. Do adversarially robust imagenet models transfer better? *Advances in Neural Information Processing Systems*, 33, 2020.

[90] S. Santurkar, A. Ilyas, D. Tsipras, L. Engstrom, B. Tran, and A. Madry. Image synthesis with a single (robust) classifier. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 1262–1273. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/6f2268bd1d3d3ebaabb04d6b5d099425-Paper.pdf`.

[91] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[92] T. Serre. Deep learning: the good, the bad, and the ugly. *Annual review of vision science*, 5: 399–426, 2019.

[93] E. Sheng, K.-W. Chang, P. Natarajan, and N. Peng. The woman worked as a babysitter: On biases in language generation. *arXiv preprint arXiv:1909.01326*, 2019.

[94] H. Song, L. Dong, W.-N. Zhang, T. Liu, and F. Wei. Clip models are few-shot learners: Empirical studies on vqa and visual entailment. *arXiv preprint arXiv:2203.07190*, 2022.

[95] S. Subramanian, W. Merrill, T. Darrell, M. Gardner, S. Singh, and A. Rohrbach. Reclip: A strong zero-shot baseline for referring expression comprehension. *arXiv preprint arXiv:2204.05991*, 2022.

[96] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014. URL `http://arxiv.org/abs/1312.6199`.

[97] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[98] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR, 2021.

[99] P. Vaibhav Rokde, Matthew Jansen. Indian birds species image classification, 2023. URL `https://www.kaggle.com/datasets/ichhadhari/indian-birds`. Dataset originally sourced from eBird, Cornell Lab of Ornithology. https://media.ebird.org/.

[100] G. Van Horn, S. Branson, R. Farrell, S. Haber, J. Barry, P. Ipeirotis, P. Perona, and S. Belongie. Building a bird recognition app and large scale dataset with citizen scientists: The fine print in fine-grained dataset collection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 595–604, 2015.

[101] G. Van Horn, E. Cole, S. Beery, K. Wilber, S. Belongie, and O. Mac Aodha. Benchmarking representation learning for natural world image collections. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12884–12893, 2021.

[102] T. Verge. What a machine learning tool that turns obama white can (and can't) tell us about ai bias - the verge. `https://www.theverge.com/21298762/face-depixelizer-ai-machine-learning-tool-pulse-stylegan-obama-bias`, 7 2022. (Accessed on 05/19/2022).

[103] Y. Vinker, E. Pajouheshgar, J. Y. Bo, R. C. Bachmann, A. H. Bermano, D. Cohen-Or, A. Zamir, and A. Shamir. Clipasso: Semantically-aware object sketching. *arXiv preprint arXiv:2202.05822*, 2022.

[104] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. 2011. URL https://authors.library.caltech.edu/27452/1/CUB_200_2011.pdf.

[105] H. Wang, Z. Wang, M. Du, F. Yang, Z. Zhang, S. Ding, P. Mardziel, and X. Hu. Score-cam: Score-weighted visual explanations for convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 24–25, 2020.

[106] J. Wang, H. Liu, X. Wang, and L. Jing. Interpretable image recognition by constructing transparent embedding space. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 895–904, 2021.

[107] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[108] Y. Xian, C. H. Lampert, B. Schiele, and Z. Akata. Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly. *IEEE transactions on pattern analysis and machine intelligence*, 41(9):2251–2265, 2018.

[109] C. Xie, M. Tan, B. Gong, J. Wang, A. L. Yuille, and Q. V. Le. Adversarial examples improve image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 819–828, 2020.

[110] M. Xue, Q. Huang, H. Zhang, L. Cheng, J. Song, M. Wu, and M. Song. Protopformer: Concentrating on prototypical parts in vision transformers for interpretable image recognition. *arXiv preprint arXiv:2208.10431*, 2022.

[111] Y. Yang, A. Panagopoulou, S. Zhou, D. Jin, C. Callison-Burch, and M. Yatskar. Language in a bottle: Language model guided concept bottlenecks for interpretable image classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19187–19197, 2023.

[112] D. Yin, R. Gontijo Lopes, J. Shlens, E. D. Cubuk, and J. Gilmer. A fourier perspective on model robustness in computer vision. *Advances in Neural Information Processing Systems*, 32:13276–13286, 2019.

[113] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, pages 3320–3328, 2014.

[114] P. Young, A. Lai, M. Hodosh, and J. Hockenmaier. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *TACL*, 2: 67–78, 2014.

[115] Q. Zhang, L. Rao, and Y. Yang. Group-cam: Group score-weighted visual explanations for deep convolutional networks. *arXiv preprint arXiv:2103.13859*, 2021.

[116] M. Zheng, P. Gao, R. Zhang, K. Li, X. Wang, H. Li, and H. Dong. End-to-end object detection with adaptive clustering transformer. 2021.

[117] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. 2016.

[118] Y. Zhu, M. Elhoseiny, B. Liu, X. Peng, and A. Elgammal. A generative adversarial approach for zero-shot learning from noisy texts. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1004–1013, 2018.

Chapter 6

Appendix

6.1 Shape and Simplicity (Chapter 2) Supplementary Materials

6.1.1 Shapeless Images

Stylized ImageNet To construct a set of stylized ImageNet images (see Fig. 2.1e), we took all ImageNet-CL images (Sec. 2.2) and changed their textures via a stylization procedure in [32], which harnesses the style transfer technique [30] to apply a random style to each ImageNet "content" image.

B&W images For all ImageNet-CL images, we used the same process described in [32] to generate silhouettes, but we did not manually select or modify the images. We used ImageMagick [44] to binarize ImageNet images into B&W images via the following steps:

```
convert image.jpeg image.bmp
potrace -svg image.bmp -o image.svg
rsvg-convert image.svg > image.jpeg
```

Silhouette For all ImageNet-CL images, we obtained their segmentation maps via a PyTorch DeepLab-v2 model [16] pre-trained on MS COCO-Stuff. We used the ImageNet-CL images that belong to a set of 16 COCO coarse classes in [32] (e.g. bird, bicycle, airplane, etc.).

When evaluating classifiers, an image is considered correctly labeled if its ImageNet predicted label is a subclass of the correct class among the 16 COCO classes (Fig. 2.1g; mapping sandpiper → bird).

6.1.2 Convolutional layers used in Network Dissection analysis

For both standard and robust models, we ran NetDissect on 5 convolutional layers in AlexNet [52], 12 in GoogLeNet [97], and 5 in ResNet-50 architectures [40]. For each layer, we use after-ReLU activations (if ReLU exists).

AlexNet layers: conv1, conv2, conv3, conv4, conv5. Refer to these names in Krizhevsky et al. [52].

GoogLeNet layers: conv1, conv2, conv3, inception3a, inception3b, inception4a, inception4b, inception4c, inception4d, inception4e, inception5a, inception5b

Refer to these names in PyTorch code `https://github.com/pytorch/vision/blob/master/torchvision/models/googlenet.py#L83-L101`.

ResNet-50 layers: conv1, layer1, layer2, layer3, layer4

Refer to these names in PyTorch code `https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py#L145-L155`).

6.1.3 Kernel smoothness visualization

We visualize the conv1 weights of the 6 models and plot them side by side to compare their kernel smoothness (Fig. 6.1). The kernels in R models a consistently smoother than its counter part. For a more straight forward visualization, we use Spearman rank correlation score to pair the similar kernels in AlexNet and AlexNet-R (Fig. 6.2).

6.1.4 Object and color detectors of AlexNet

We compared the layer-wise difference of object and color detectors of AlexNet and AlexNet-R (Fig. 6.3). We can see in Fig. 6.3a that AlexNet has more object detectors and fewer color detectors compared to AlexNet-R.

6.1.5 Total variance (TV) of AlexNet & AlexNet-R on clean/noisy images

The total variances of AlexNet & AlexNet-R plot (Fig. 6.4) shows that the early layer i.e. conv1 in AlexNet-R can filtered out Gaussian noise. The total variances of clean and noisy input results in smilar value in conv1 in AlexNet-R, this means the noise has been filtered by conv1.

6.1.6 ImageNet-C evaluation

In Table 6.1, we evaluate the validation accuracy of 6 models on 15 common types of image corruptions in ImageNet-C. It turns out that shape biased model does not necessary mean better generalization on these image corruptions.

Table 6.1: Top-1 accuracy of 6 models (in %) on all 15 types of image corruptions in ImageNet-C [41]. On average over all 15 distortion types, R models underperform their standard counterparts.

		AlexNet		GoogLeNet		ResNet			
		Standard	Robust	Standard	Robust	Standard	Robust	adv-prop PGD1	adv-prop PGD5
(a) Noise	Gaussian	11.05	**56.10**	**56.34**	26.78	**38.43**	45.03	49.53	**56.39**
	Shot	11.15	**53.22**	**50.43**	24.66	35.03	**44.07**	47.56	**52.85**
	Impulse	8.93	**52.49**	**42.10**	24.90	38.40	**41.68**	49.53	**52.39**
(b) Blur	Defocus	24.34	**28.15**	**39.63**	26.52	**43.77**	37.95	49.71	**56.06**
	Glass	22.76	**44.50**	22.62	**43.17**	20.71	**51.53**	30.38	**37.74**
	Motion	33.87	**41.74**	**44.24**	41.25	44.92	**50.16**	48.27	**54.63**
	Zoom	38.86	**44.27**	42.58	**43.61**	45.16	**52.20**	49.59	54.97
(c) Weather	Snow	26.59	**26.91**	**52.78**	15.60	**42.13**	40.39	46.01	**51.46**
	Frost	**21.46**	13.85	**46.14**	8.38	**37.72**	33.51	43.88	**50.66**
	Fog	**27.86**	1.64	**64.37**	12.30	**56.78**	3.81	54.81	**58.64**
	Brightness	**77.61**	62.67	**91.60**	51.14	**85.67**	79.44	86.76	**88.30**
(d) Digital	Contrast	**18.24**	2.06	**78.38**	22.63	**53.67**	3.46	56.57	**61.63**
	Elastic	75.97	**80.98**	**78.18**	77.15	67.86	**82.11**	74.23	**78.44**
	Pixelate	57.94	**79.46**	**82.47**	79.35	62.40	**83.21**	67.88	**76.38**
	JPEG	72.82	**85.07**	80.27	**81.72**	73.66	**85.51**	79.75	**82.01**
(e) Extra	Speckle Noise	17.55	**58.42**	**51.32**	31.31	41.74	**52.57**	52.80	**58.07**
	Gaussian Blur	28.68	31.26	**45.52**	30.36	**47.56**	41.70	55.68	**60.43**
	Spatter	28.68	31.26	**45.52**	30.36	**47.56**	41.70	55.68	**60.43**
	Saturate	46.90	**63.66**	**65.36**	57.48	58.58	**70.72**	66.21	**71.44**
mean Accuracy		37.16	**47.34**	**59.38**	40.34	51.70	51.72	57.86	**62.80**

6.1.7 Examples of shape-less and texture-less images

We randomly choose one image in 7 COCO coarser classes (out of 16) and plot the corresponding shape-less and texture-less version (Fig. 6.5).

6.1.8 Visualizing channel preference via cue-conflict and NetDissect

In Fig. 6.6- 6.8, **Top** is the top-49 images of the channels (Similar to Fig. 2.7). On the **Middle & Bottom**, we zero-out the corresponding channel and re-run the conflict test to find out the images that were mis-classified. i.e. Fig.6.6 the clock images in **Middle** were classified into shape category by cue-conflict test. After zeroing-out the channel, the network lose the ability to classify the image into shape category.

6.2 gScoreCAM (Chapter 3) supplementary materials

6.2.1 Experiments & Findings

In order to directly compare with performance, we use weakly supervised object localization (WSL) to evaluate the heatmaps generated by different methods in Section 6.2.4. This evaluation gives information on how accurate the generated heatmap is with respect to the target. In our experiments , We find that gScoreCAM is the best method in COCO and PartImageNet (see Table 3.2 for details).

In this section, we first introduce some ablation studies (Section 6.2.2) to explain the choice of hyperparameter k and gradient dimension reduction. Then, we study why gScoreCAM is better than other CAM-based methods in Section 6.2.3. With the choice of our design, we conduct a systematic WSL evaluation

in Section 6.2.4. To better understand these WSL results, we further study how they perform in different scenarios, e.g., object size and number of objects, in Section 6.2.5. Lastly, we conduct the same experiment as in Section 6.2.4 but for a different model (RN50x4), which shows that our method is consistently better than others.

6.2.2 Ablation study of gScoreCAM

We first introduce why we set $k = 300$ in our proposed method by studying how the hyperparameter k affects the WSL performance. We then compare WSL performance using different pooling methods to rank the channels.

Effect of number of channels

From Table 6.2, we see that gScoreCAM reaches its peak performance as the number of channels increases to 500. We choose $k = 300$ to conduct most of our experiments since it has similar performance to $k = 500$ but runs much faster (at only 60% of the run-time).

Table 6.2: k = 300 is the smallest number of channels that yield a high localization accuracy on ImageNet-v2 and COCO.

Number of channels	ImageNet-v2 [84] (MaxBoxAcc)	COCO [56] (BoxAcc)
300 (random)	55.12	18.55
20	49.83	15.72
100	54.97	19.25
200	53.75	20.50
300	**56.61**	**20.83**
400	56.60	20.89
500	**57.38**	**20.89**
600	56.55	20.89

Table 6.3: Taking the mean of the channel-wise gradients yields the highest localization accuracy when k = 300.

	ImageNet-v2 [84] (MaxBoxAcc)	COCO [56] (BoxAcc)
Mean	**56.61**	**20.83**
Max pooling	56.46	20.62
Average pooling	54.57	20.35

Effects on the choice of gradient dimension reduction

To use the gradient to guide us in choosing important channels, we need to reduce the dimensions of the gradient from $\mathbb{R}^{c \times w \times h}$ to $\mathbb{R}^{c}$. Here, we study some of the most common methods, mean, Max-Pooling, and Average-Pooling on gScoreCAM. As the result shows in Table 6.3, the best method is the simple mean value. This result coincides interestingly with the choice of GradCAM [91].

6.2.3 Why gScoreCAM is better in weighting the activation maps?

This section aims to examine weights quality by measuring levels of overlap between the target and activation map. Therefore, we design an experiment that directly measures the quality of the weights used by different methods. We compare methods that directly weigh the activation maps (GradCAM, xGradCAM, GradCAM++, ScoreCAM, and gScoreCAM) and find that gScoreCAM is the best among them. We also measure the noise level of the weighted activation map (Table 6.4) with Total Variation [81]. Note that the noise level can not directly reflect the performance of the object localization but instead indicates the *confidence* of the visualizing method.

gScoreCAM is the best weighting system among the candidates

Experiment We design an experiment to measure the quality of the weighting systems by directly measuring the level of overlap of the weighted activation maps and the ground truth. More specifically, we first measure the area under the curve (AUC) of the IoU over different binary thresholds (e.g., $\tau \in [0.0 : 0.05 : 0.95]$) for each activation map that CAM-based methods use. We then compute the weighted sum of the corresponding AUC with the weights given by the method. Finally, we average the weighted AUC of all testing samples. Note that in some methods (GradCAM, xGradCAM, GradCAM++) in which the weights are not summed to one, we first discard the negative values and divide the remaining by its sum. This will provide us the upper bound of the mean weighted AUC. In this experiment, we use 4000 object-image pairs with 50 samples per class in the COCO training set. We set a baseline by simply assuming that all activation maps have equal weights.

Results We find that the gScoreCAM weighting is 2x better than the upper bound of GradCAM and slightly better than ScoreCAM (Table 6.4). The upper bound of the mean weighted AUC of GradCAM (in ImageNet-v2) is actually similar to the baseline, which explains why the performance of GradCAM is worse than the Gaussian noise baseline in [19]. Another interesting fact is that, on average, the best IoU of the activation maps is about 0.76. This indicates that the model indeed leverages the correct information (target object) during the process most of the time. Note that we are not sure how the model will utilize this information; this just reminds us that the image encoder is capable of finding the target.

Table 6.4: We directly evaluate the weighting of different CAM-based methods and found that gScoreCAM is the best among them. The uniform baseline is simply averaging all the activation maps. The total variation of the heatmap is generated by different methods. It gives us information about how noisy the heatmap is. The lower Total Variation means that the heatmap tends to be less noisy.

	Mean weighted AUC	Mean Total Variation
Baseline	0.0380	1745 ± 268
GradCAM [91]	0.0390	885 ± 484
xGradCAM [28]	0.0421	1090 ± 657
GradCAM++ [11]	0.0357	1500 ± 458
ScoreCAM [105]	0.0881	1363 ± 391
gScoreCAM (ours)	**0.0936**	1301 ± 422

gScoreCAM is less noisy compared to ScoreCAM

Experiment We use 10,000 randomly sampled object-image pairs from the COCO validation set in this experiment. We compute the mean total variation of the heatmaps generated by different methods.

Results The total variation gives us a statistical view of the noise level of the resulting heatmap from different methods. We can see that gScoreCAM is statistically less noisy than ScoreCAM in Table 6.4. An interesting result is that GradCAM is the least noisy method. Although we do want the resulting heatmap to be less noisy, the level of noise itself can not guarantee better performance.

Why GradCAM is the least noisy method? As discussed above, GradCAM tends to give a much cleaner and lower coverage heatmap compared to other methods. We study the weighted activation maps of GradCAM over 4,000 random samples and find that the average resulting heatmap is entirely negative. Statistically, about 47% of the weights in GradCAM are negative (averaged in 4,000 samples). On the other hand, the weights of gScoreCAM and ScoreCAM are all positive, which leads to higher Total Variation and hence, a nosier heatmap. Note that the activation maps are after ReLU; therefore, all the activation maps are positive.

6.2.4 Zero-shot Object localization

We evaluate different visualization methods based on their zero-shot object localization performance. Thanks to CLIP, we are able to directly test the object/part localization in different datasets without fine-tuning. Since the CAM-based methods aim to give the corresponding area of a class/prompt, measuring the performance of localization will be a direct measurement of the visualization results.

Evaluation Results

ScoreCAM and gScoreCAM are the best methods among the tests From the object localization test on ImageNet-v2, COCO, and PartImageNet in Table 3.2, we can see that ScoreCAM and gScoreCAM consistently outperform other methods. In particular, they are about 1.5-4 times better in COCO and 1.2-2 times better in PartImageNet compared to other methods.

gScoreCAM runs 8 times faster than ScoreCAM Since the visualization methods are model dependent, we will measure the run-time by its approximate run-time and the forward/backward passes required by these methods. We measure the average run time (in seconds) for each image-prompt pair under different CAM methods and report in Table 3.2. The average run-time is measured by averaging the total time of 200 samples on a single Nvidia 1080Ti GPU.

The required forward pass of ScoreCAM is 10 times more than gScoreCAM, which means that theoretically it needs 10 times more computation and results in 10 times longer run-time. In our approximate experiments, the actual run-time of gScoreCAM is about 8 times less than ScoreCAM.

gScoreCAM performs better on complex tasks The object localization task on ImageNet-v2 is considered relatively "simple" because most of the test images are object-centric and a center Gaussian blur could reach 52.5% accuracy as reported in [19]. However, the same task on COCO and PartImageNet is much harder due to the variety of target sizes, shapes, and locations. Interestingly, gScoreCAM performs better on these harder tasks than ScoreCAM.

Apply gScoreCAM to ViT-based CLIP Although gScoreCAM is designed for the CNN-based model, we can also apply it to the ViT-based model by reshaping the embedding as we discussed in Section 3.3.2. As shown in Table 3.2, we can achieve a similar performance compared to HilaCAM, which is the state-of-the-art method to visualize ViT. One interesting note is that HilaCAM only pays attention to generating heat maps, and our method only uses activation. Despite the enormous differences in the approaches, both techniques end up giving similar results.

6.2.5 Why does gScoreCAM perform better in COCO and PartImageNet?

From Table 3.2 we find that gScoreCAM is slightly worse in ImageNet-v2 but better in COCO and PartImageNet compared to ScoreCAM. In this section, we conduct two sets of controlled experiments to find out why gScoreCAM is better in these two datasets.

gScoreCAM performs better when there are more objects in an image

Experiment We conduct a controlled experiment based on the number of objects in an image. Specifically, we evaluate the performance of different methods when the number of classes (one object per class) are different in the image. For diversity, we select images that each class only have one instance. We use a union of the COCO and LVIS labels in this experiment because we can have a more accurate label for each image. We directly measure the mean IoU in each group using the same method as in Section 6.2.4. We test the number of classes from 1-9 in an image and split them into three groups, 1-3 , 4-6, and 7-9 classes with 1150, 2790, and 874 samples correspondingly.

Results We find that as the number of classes per image increases, the IoU of all methods decreases. gScoreCAM has the lowest IoU drop among them, resulting in better performance when the number of classes per image is greater than 4. The median IoU of gScoreCAM is approximately 0.03 higher than ScoreCAM and 0.07 to 0.10 higher than GradCAM and HilaCAM, as shown in Figure 6.13a. This advantage makes gScoreCAM perform better in COCO given that COCO is a dataset that often has multiple objects in an image.

gScoreCAM can better localize small or part of the object

Experiment Similar to Section 6.2.5, we conduct another controlled experiment on COCO and PartImagenet based on the size of the target part/object. In this experiment, we divide the test samples into three groups according to the target ratio. The target ratio is measured by the area of the target part/object over the area of the full image. We simply divide images into three groups by the target ratio: small (0-0.33), medium (0.33-0.67), and large (0.67-1). Where COCO and PartImageNet have (13811, 607, 210) and (12727, 1222, 307) samples in the corresponding group.

Results As Figures 6.13b and 6.13c show that gScoreCAM consistently has a higher IoU in the small object groups, while ScoreCAM always has a higher IoU in the large groups. Combined with the results in Section 6.2.3, we find that ScoreCAM tends to generate a large and possibly noisy heatmap, resulting in better performance in the WSL task when the object is large. On the other hand, gScoreCAM can better locate small objects or parts. This capability is important because, in terms of explainability, we would like the resulting heatmap to be as accurate as possible.

6.2.6 gScoreCAM consistently better on different CLIP models

We repeat the WSOL experiments in Section 6.2.4 for RN50x4 to confirm that our proposed method can generally be applied to different models. For generality (i.e. no further hyperparameter tuning), we use the same hyperparameters $k = 300$ as in Section 6.2.4.

Results Our proposed method is consistently the best in terms of zero-shot localization. As shown in Table 6.5, the accuracy of gScoreCAM is around 2 to 4× higher than other methods (except ScoreCAM). It suggests that gScoreCAM could be applied to other variations of CLIP without hyperparameter tuning.

Table 6.5: For both CLIP RN50x16 and RN50x4, gScoreCAM is the most accurate localization method in the CAM-based family on ImageNet-v2, COCO, and PartImageNet.

	ImageNet-v2 [84]		COCO [56]		PartImageNet [39]	
	RN50x16	RN50x4	RN50x16	RN50x4	RN50x16	RN50x4
GradCAM [91]	38.9	32.61	11.59	9.86	10.91	9.60
xGradCAM [28]	24.24	18.94	5.6	6.11	6.57	5.01
GradCAM++ [11]	44.15	46.23	9.68	10.68	2.93	8.00
LayerCAM [46]	43.7	47.01	9.19	9.87	12.42	13.36
GroupCAM [115]	50.85	22.77	13.06	1.29	6.16	3.01
ScoreCAM [105]	**57.78**	57.99	20.43	21.31	15.67	15.39
gScoreCAM (ours)	56.61	**58.76**	**20.83**	**22.17**	**16.34**	**16.22**

6.2.7 Qualitative study via CLIP

CLIP had been shown to have a high recall in zero-shot cross-modal image retrieval [80] in Flickr30K [114] and COCO [56]. In this section, we study the heatmaps given by CLIP with different visualization methods. We first show a progressing plot that shows how the heatmap changes with the number of channels used by gScoreCAM. We then visually compare the heatmap from different methods. From the visual studies, we find that our purposed gScoreCAM gives a more accessible explanation of the model.

The heatmap is getting nosier as the number of channels used by gScoreCAM increasing

Figure 3.1 shows a progressing plot when the number of channels used by gScoreCAM increases from 300 to 3072 (ScoreCAM) and the heatmap generated by RISE (last column). We can see a clear trend, which shows that the gradient-guided ranking successfully ranks the channels by the heatmaps' contribution to the target. This visualization further confirms the result in Section 6.2.2 that we only need the top k channels to localize the target.

Visual comparison of gScoreCAM to other methods

We study a series of visualization using different methods (see Figures 6.18 to 6.24). It turns out that when the target object is very small (e.g., fish tail), gScoreCAM can generate a very accurate heatmap, while other methods usually result in nosier or incorrect heatmaps, as shown in Figure 6.14. These faithful heatmaps would allow us to study what the model is really looking at, and hence can be a useful tool to study the model. See Figures 6.16 and 6.17 for more comparison between gScoreCAM and other methods.

6.2.8 Derive bounding box from heatmap

Get bounding box with Otsu's method

figure 6.15 shows the intermediate results during the procedure (prompt: dog). As described in Algorithm 3, we obtain the bounding box using the following procedure:

1. From an input image, we used a CAM method to get a heatmap.

2. Binarize the heatmap with Otsu's method.

3. Find the contours of the binary map (using the OpenCV function $cv2.findContours$).

4. For each contour, determine a minimal bounding box (using the OpenCV function $cv2.boundingRect$).

5. Then choose the largest bounding box as result.

Algorithm 3 Derive Bounding Box from Heatmap

```
Input: (i) A heatmap $M \in \mathbb{R}^{u \times v}$ from any CAM methods. (ii) Input image size $(w, h)$.
Output: A bounding box for the target class or prompt.
$M \leftarrow$ Upsample($M$, size = $(w, h)$, method = "bilinear");
$M \leftarrow$ Otsu($M$);
$contours \leftarrow$ findContours($M$);
$boxes \leftarrow [[0,0,0,0], \dots, [0,0,0,0]]$;
for $i = 0$ to $len(contours) - 1$ do
    $boxes[i] \leftarrow$ boundingRect($contours[i]$);
end
$u \leftarrow \arg\max_i$ Area($boxes[i]$);
$output \leftarrow boxes[u]$;
```

Effects on using Otsu's method

In this section, we study how Otsu's method differs from the commonly used single threshold by grid search. We performed the experiment on the COCO validation set. We keep the other procedure the same as in Algorithm 3, except that we replace Otsu's binarization with using a single threshold. We perform a grid search for the optimal threshold on the subset of images in the training set (100 images per class) for each method with $gridsize = 0.05$. The search is based on the mean IoU over the search samples. We found that the difference between using Otsu's method and using optimal threshold by grid search is trivial.

Table 6.6: Difference between Otsu binarzation with optimal thesholding. The difference between Otsu's method and using optimal threshold value is very small.

	Single value	Otsu	Difference
GradCAM	11.95%	11.56%	-0.39%
GradCAM++	9.03%	9.68%	0.65%
xGradCAM	6.85%	5.60%	-1.25%
GroupCAM	5.58%	4.52%	-1.06%
LayerCAM	9.99%	9.19%	-0.80%
ScoreCAM	20.74%	20.43%	-0.31%
gScoreCAM	20.27%	20.83%	0.56%
Hila' method(ViT-B/32)	13.33%	12.82%	-0.51%
ScoreCAM(ViT-B/32)	9.60%	10.21%	0.61%
gScoreCAM(ViT-B/32)	9.40%	10.10%	0.70%
Mean	11.67%	11.49%	-0.18%

6.2.9 Visualizations of different methods

In this section, we show a series of comparisons between different methods. We found that for these hard tasks (figure 6.18), gScoreCAM gives better performance in COCO and PartImageNet.

Advantages of gScoreCAM

We show a few sample visualizations of gScoreCAM, GradCAM, ScoreCAM and HilaCAM in figures 6.17 and 6.18.

6.3 PEEB (Chapter 4) Supplementary Materials

6.3.1 Architecture details

Image encoder and text encoder

We employ the image encoder and text encoder from OWL-ViT. In order to maintain a general understanding of natural languages and avoid overfitting our training samples, we keep the text encoder frozen for all training and experiments. This setup allows our design to be flexible about the choice of text encoder, e.g., one can easily replace the text encoder without changing other architecture.

Linear projection (for part embedding selection)

The image embedding will be forwarded to a **Linear Projection** layer (see detail implementation here), which is composed of a learnable logit scale, a learnable logit shift, and an Exponential Linear Unit (ELU) activation function. These processed image embeddings then have the same dimension as the text embeddings. For OWL-ViT$_{\text{base32}}$, the image embeddings are projected from 768 to 512. We select a single image embedding for each text query. In this context, the text queries correspond to the component names of the target object, which includes twelve distinct parts. This selection is based on the cosine similarity between the projected image embeddings and the text embeddings. Finally, the chosen images embeddings (before projection) will be sent to the **Part MLP** for classification and **Box MLP** for box prediction (figure 6.25, Step 1).

Part MLP

We introduce **Part MLP** to enable part-based classification (see implementation detail here). It comprises a three-layer MLP with GELU activations [42] . **Part MLP** takes in the selected part embeddings (i.e. output of step 1 in figure 6.25) and outputs a vector of size $\mathbb{R}^d$ for each part, where d is the dimension of descriptor embeddings (for OWL-ViT$_{\text{base32}}$, the input dimension is 768, and $d = 512$). Part MLP is trained to map the selected part embeddings to the same dimensional space with descriptor embeddings to compute final logits for classification.

Box MLP

The **Box MLP** retained from OWL-ViT consists of a three-layer MLP (see here for implementation detail). It takes the visual embedding as input and generates a four-element vector corresponding to the center coordinates and size of a bounding box (e.g., `[x, y, width, height]`). It is important to note that the image embedding inputs of **Box MLP** and **Part MLP** layers are the same, as shown in figure 6.25, Step 2.

Visual part embedding selection

As shown in figure 6.25 step 1, 1c, the image embeddings are first projected by a Linear Projection layer and compute the dot product with the encoded part names. The image embeddings (before Linear Projection) are chosen as visual part embeddings by selecting the embedding that has the highest similarity scores with the corresponding part after the Linear Projection.

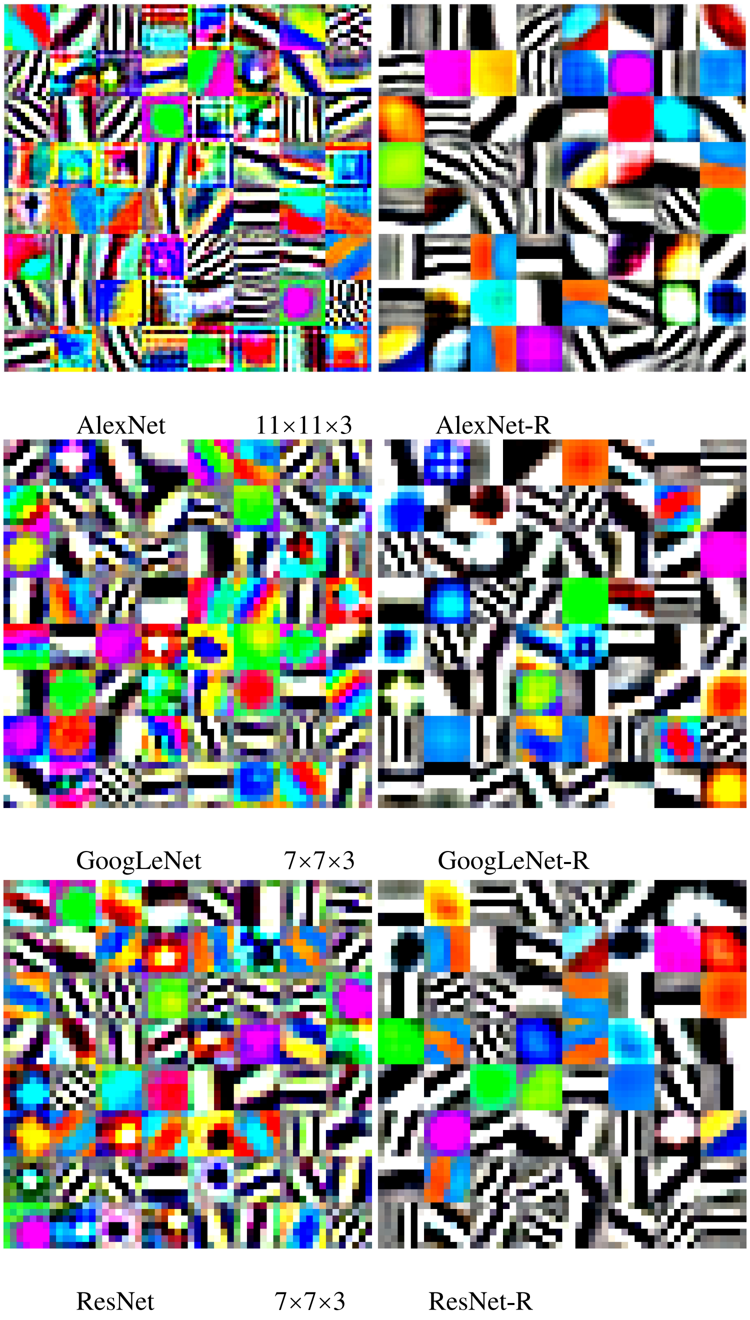


Figure 6.1: All 64 conv1 filters of in each standard network (left) and its counterpart (right). The filters of R models (right) are smoother and less diverse compared to those in standard models (left). Especially, the edge filters of standard networks are noisier and often contain multiple colors in them.

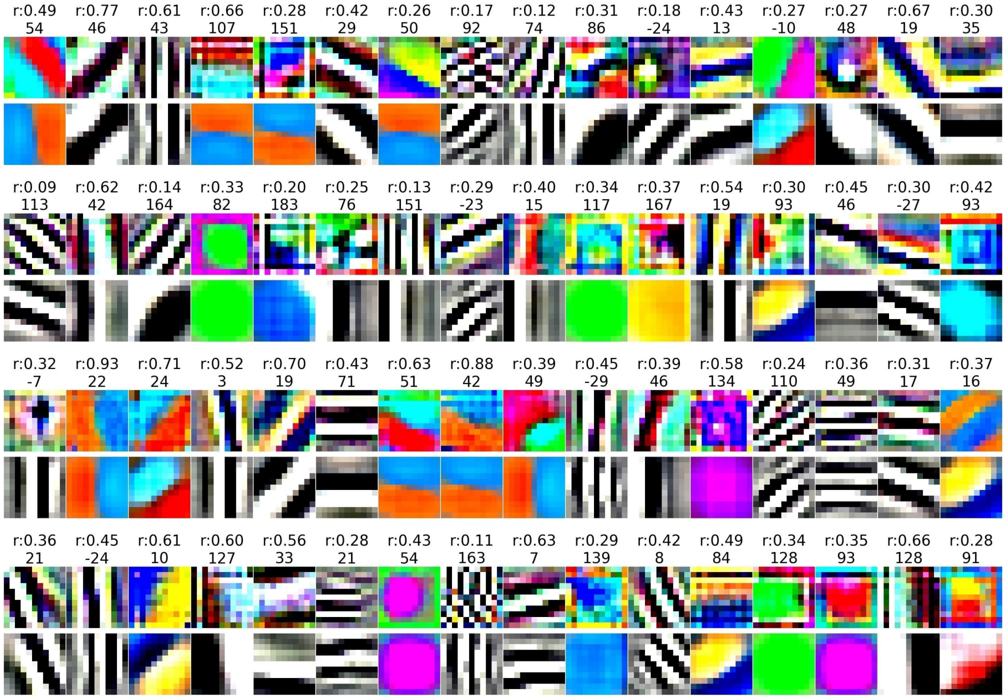
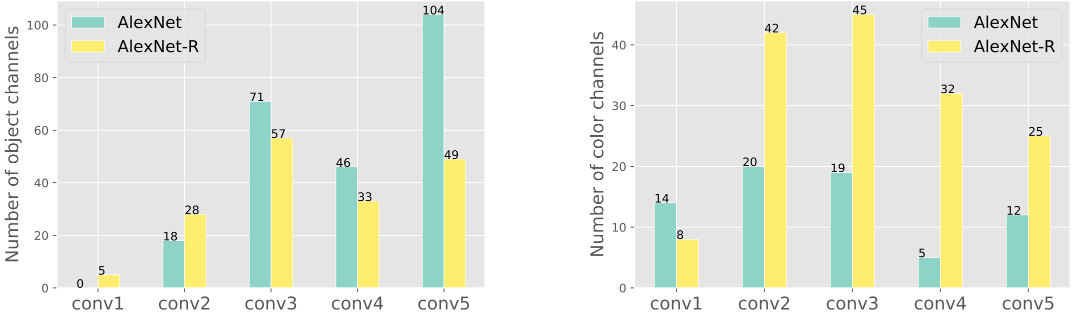
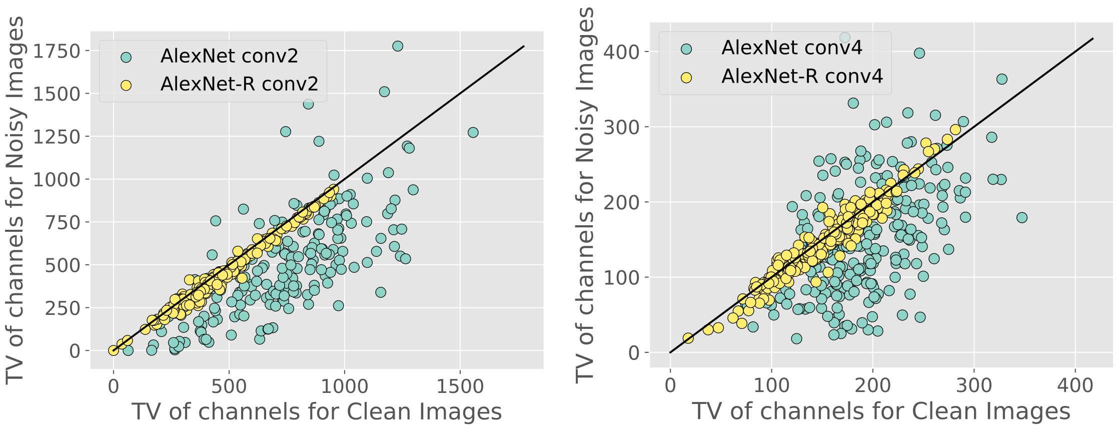


Figure 6.2: conv1 filters of AlexNet-R are smoother than the filters in standard AlexNet. In each column, we show an AlexNet filter conv1 filter and their nearest filter (bottom) from the AlexNet-R. Above each pair of filters are their Spearman rank correlation score (e.g. r: 0.36) and their total variation (TV) difference (i.e. smoothness differences). Standard AlexNet filters are mostly noisier than their nearest R filter (i.e. positive TV differences).



(a) Number of object detectors per AlexNet layer (b) Number of color detectors per AlexNet layer

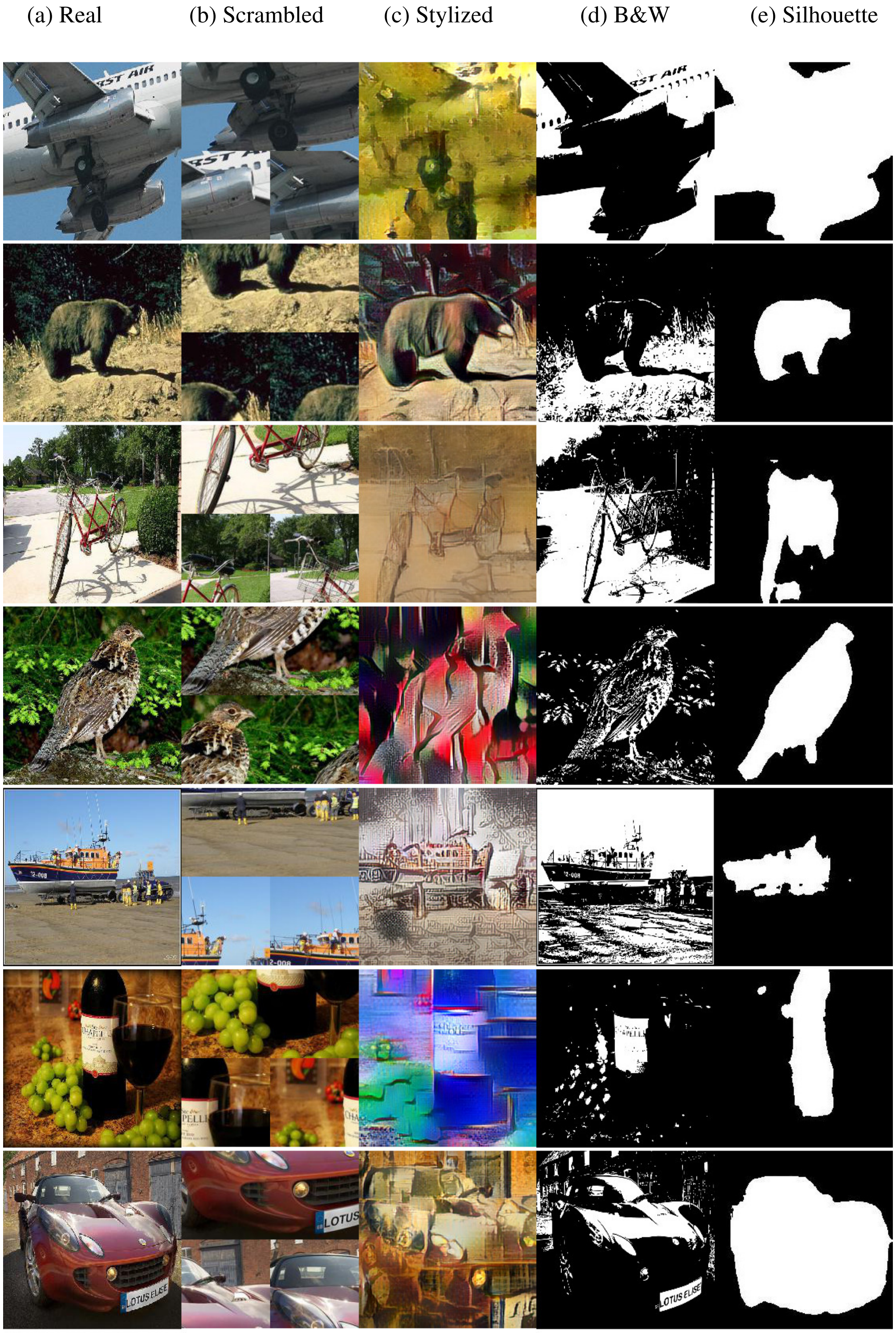
Figure 6.3: In higher layers (here, conv4 and conv5), AlexNet-R have fewer object detectors but more color detector units compared to standard AlexNet. The differences between the two networks increase as we go from lower to higher layers. Because both networks share an identical architecture, the plots here demonstrate a substantial shift in the functionality of the neurons as the result of adversarial training—detecting more colors and textures and fewer objects. Similar trends were also observed between standard and R models of GoogLeNet and ResNet-50 architectures.



(a) conv2

(b) conv4

Figure 6.4: Each point shows the Total Variation (TV) of the activation maps on clean and noisy images for an AlexNet or AlexNet-R channel. We observe a striking difference in conv1: The smoothness of R channels remains unchanged before and after noise addition, explaining their superior performance in classifying noisy images. While the channel smoothness differences (between two networks) are gradually smaller in higher layers, we still observe R channels are consistently smoother.



(a) Real (b) Scrambled (c) Stylized (d) B&W (e) Silhouette

Figure 6.5: Applying different transformation that remove shape/texture on real images. We randomly show an example of 7 out of 16 COCO coarser classes. See Table 2.3 for classification accuracy scores on different images distortion dataset in 1000 classes(Except for Silhouette). **Note: Silhouette are validate in 16 COCO coarse classes.*

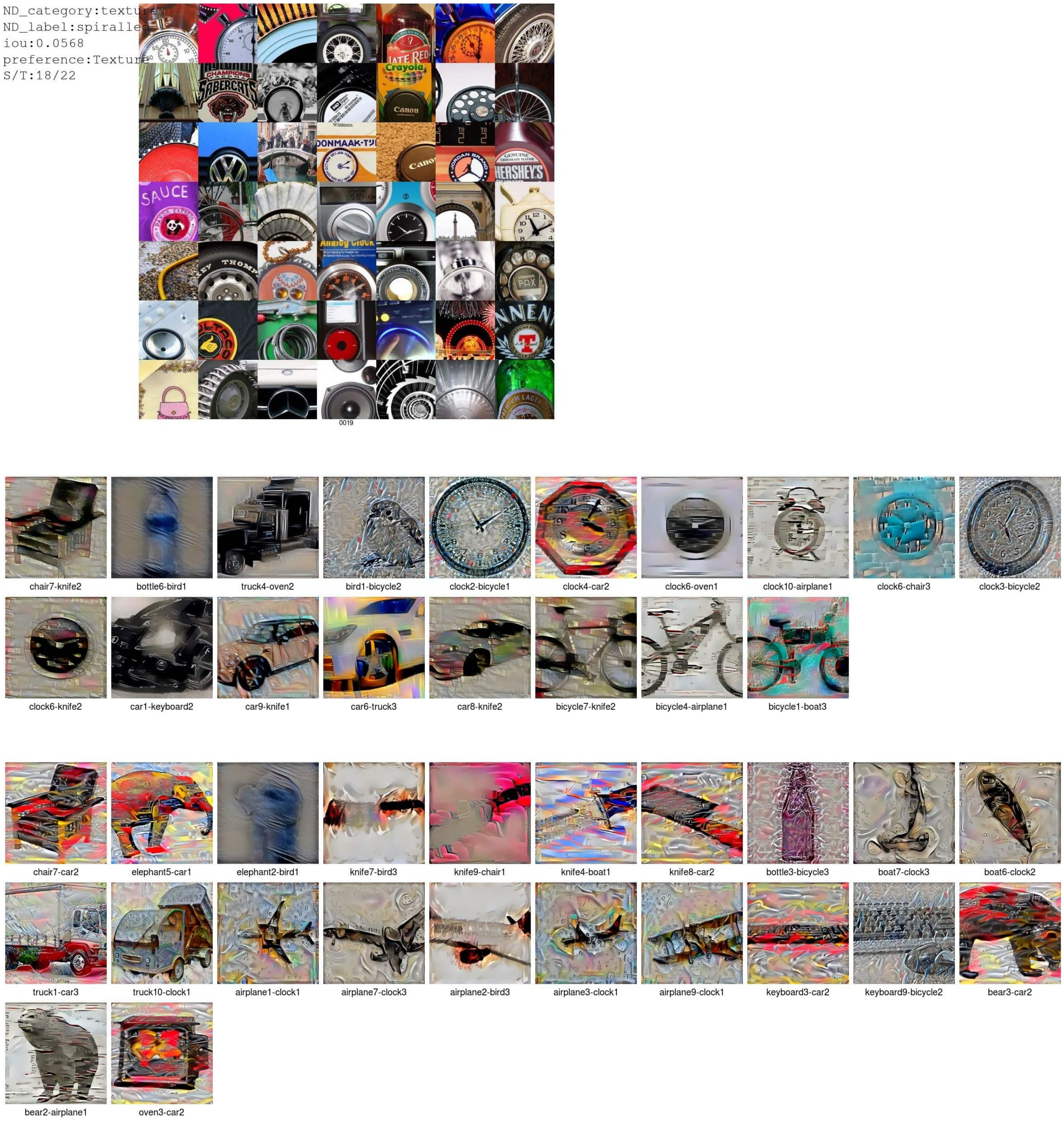


Figure 6.6: AlexNet conv4$_{19}$ with Shape and Texture scores of 18 and 22, respectively. It has a NetDissect label of spiralled (IoU: 0.0568) under texture category. Although this neuron is in NetDissect texture category, the misclassified images suggest that this neuron helps in both shape- and texture-based recognition. **Top:** Top-49 images that highest-activated this channel. **Middle:** Mis-classified images in shape category (18 images). **Bottom:** Mis-classified images in texture category (22 images).

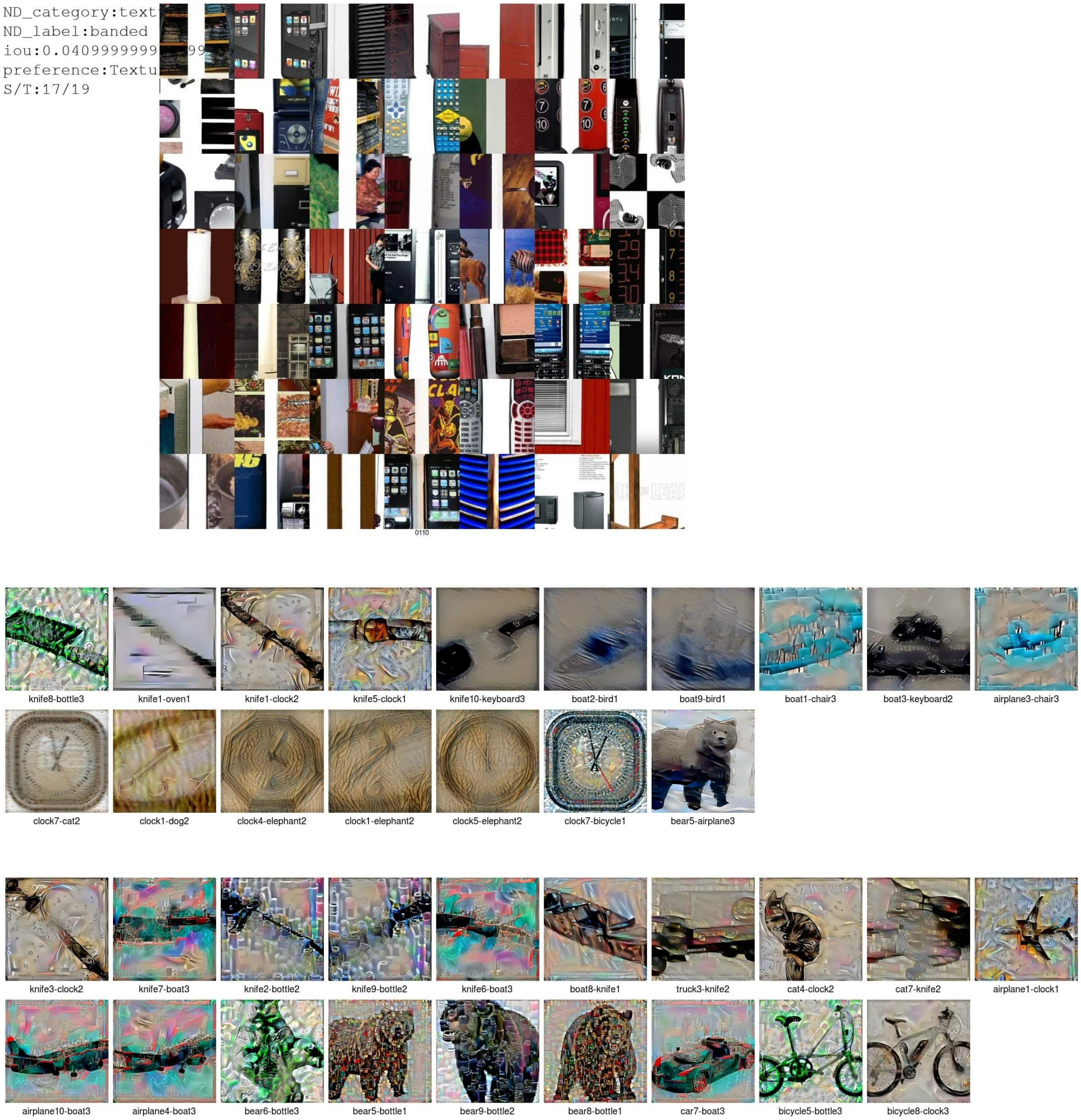


Figure 6.7: AlexNet-R conv5$_{110}$ with Shape and Texture scores of 17 and 19, respectively. It has a NetDissect label of banded (IoU: 0.0409) under texture category. This neuron has almost equal Shape and Texture scores and is useful in detecting both the shape and textures of knives and bottles at the same time. **Top:** Top-49 images that highest-activated this channel. **Middle:** Mis-classified images in shape category. **Bottom:** Mis-classified images in texture category.

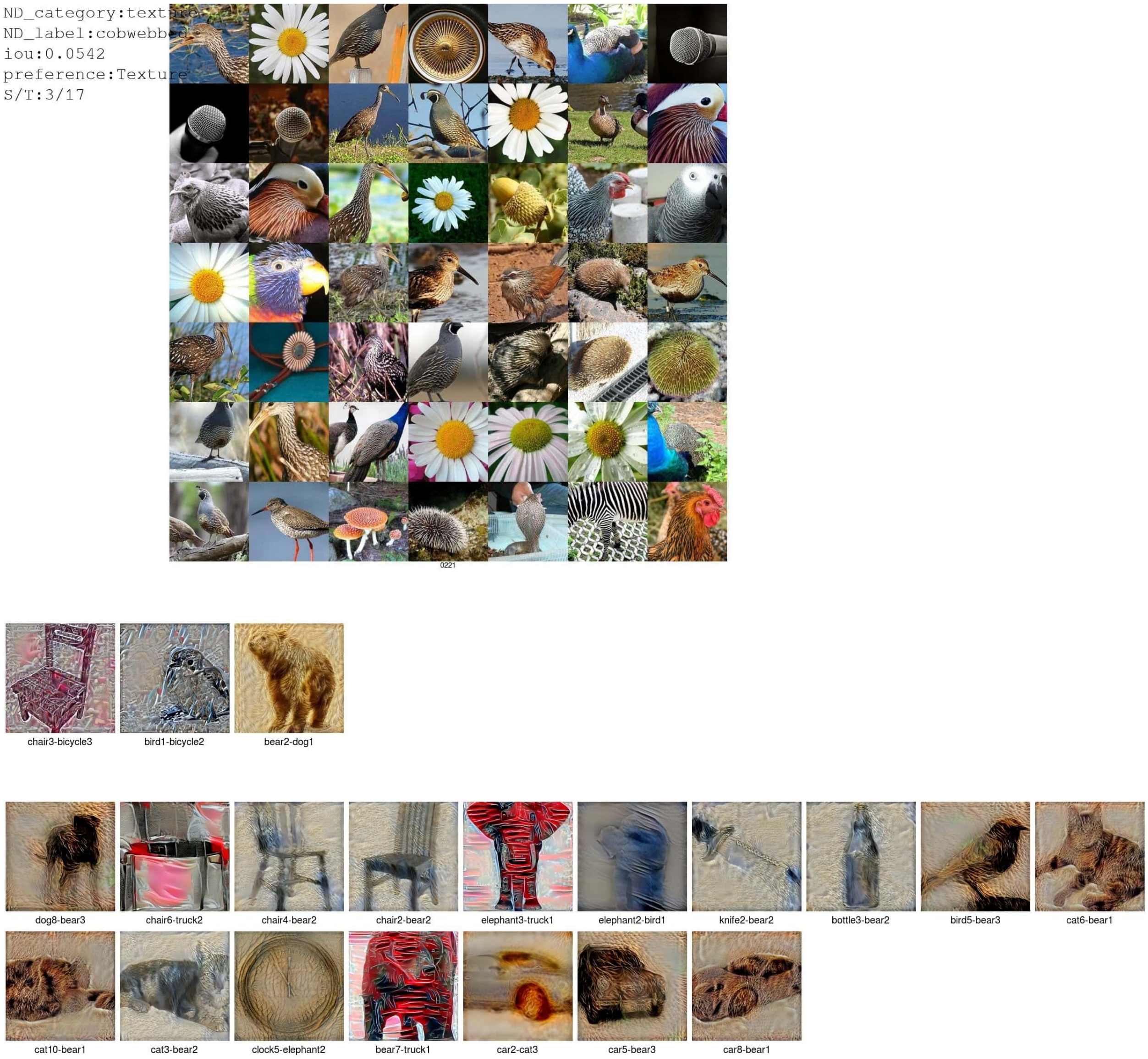
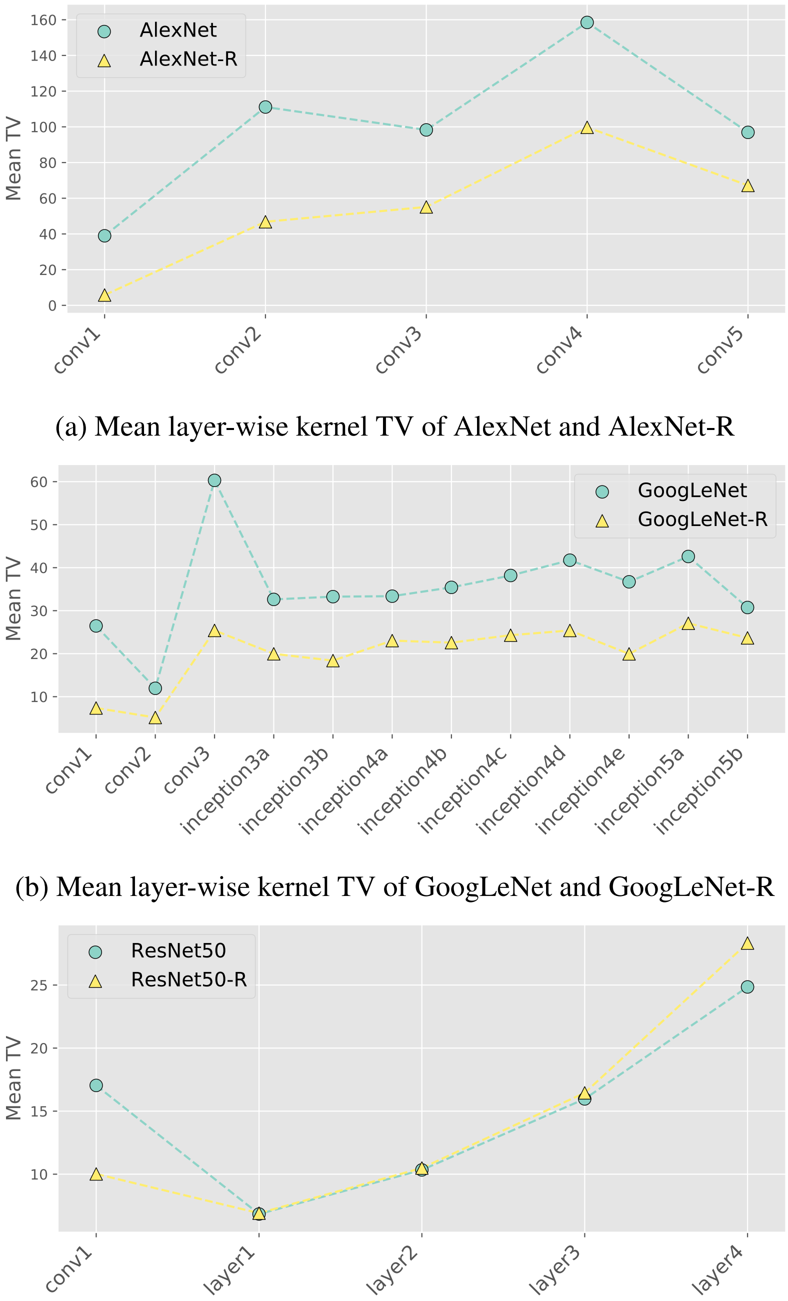


Figure 6.8: AlexNet conv5$_{221}$ with Shape and Texture scores of 3 and 17, respectively. It has a NetDissect label of cobwebbed (IoU: 0.0542) under texture category. This is a heavily texture-biased neuron that helps networks detect animals by their fur textures. **Top:** Top-49 images that highest-activated this channel. **Middle:** Mis-classified images in shape category. **Bottom:** Mis-classified images in texture category.

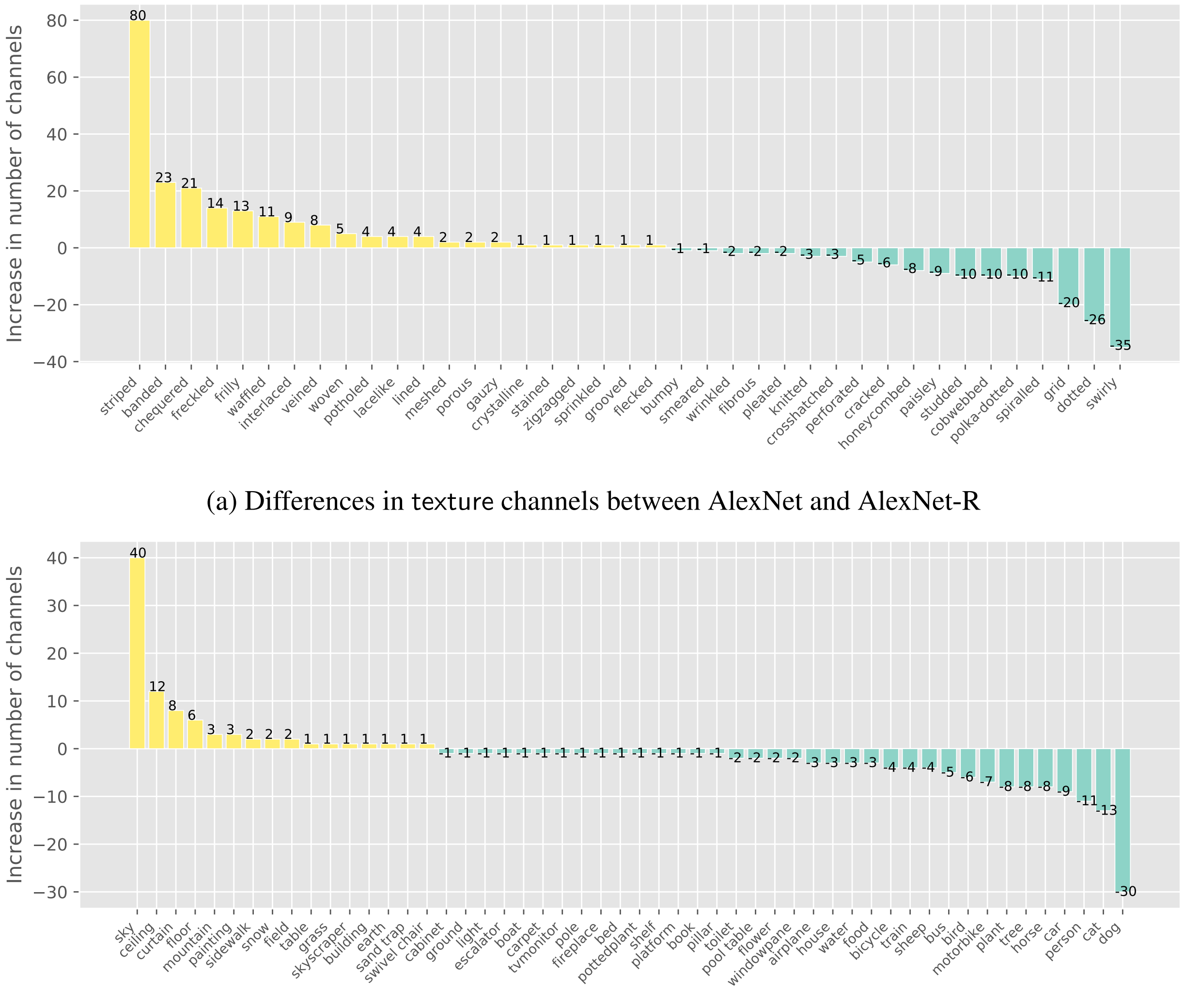


(a) Mean layer-wise kernel TV of AlexNet and AlexNet-R

(b) Mean layer-wise kernel TV of GoogLeNet and GoogLeNet-R

(c) Mean layer-wise kernel TV of ResNet and ResNet-R

Figure 6.9: For all main conv layers, AlexNet-R filters are smoother (i.e. lower TV mean) than their counterparts in AlexNet (a). The same observation was found in GoogLeNet-R vs. GoogLeNet comparison (b). In ResNet-R, its early filters at conv1 are also smoother than those in ResNet.



(a) Differences in texture channels between AlexNet and AlexNet-R

(b) Differences in object channels between AlexNet and AlexNet-R

Figure 6.10: In each bar plot, we column shows the difference in the number of channels (between AlexNet-R and AlexNet) for a given concept e.g. striped or banded. That is, yellow bars (i.e. positive numbers) show the count of channels that the R model has more than the standard network in the same concept. Vice versa, teal bars represent the concepts that R models have fewer channels. The NetDissect concept names are given in the x-axis.
Top: In the texture category, the R model has a lot more simple texture patterns e.g. striped and banded (see Fig. 6.11 for example patterns in these concepts).
Bottom: In the object category, AlexNet-R often prefers simpler-object detectors e.g. sky or ceiling (Fig. 6.10b; leftmost) while the standard network has more complex-objects detectors e.g. dog and cat (Fig. 6.10b; rightmost).

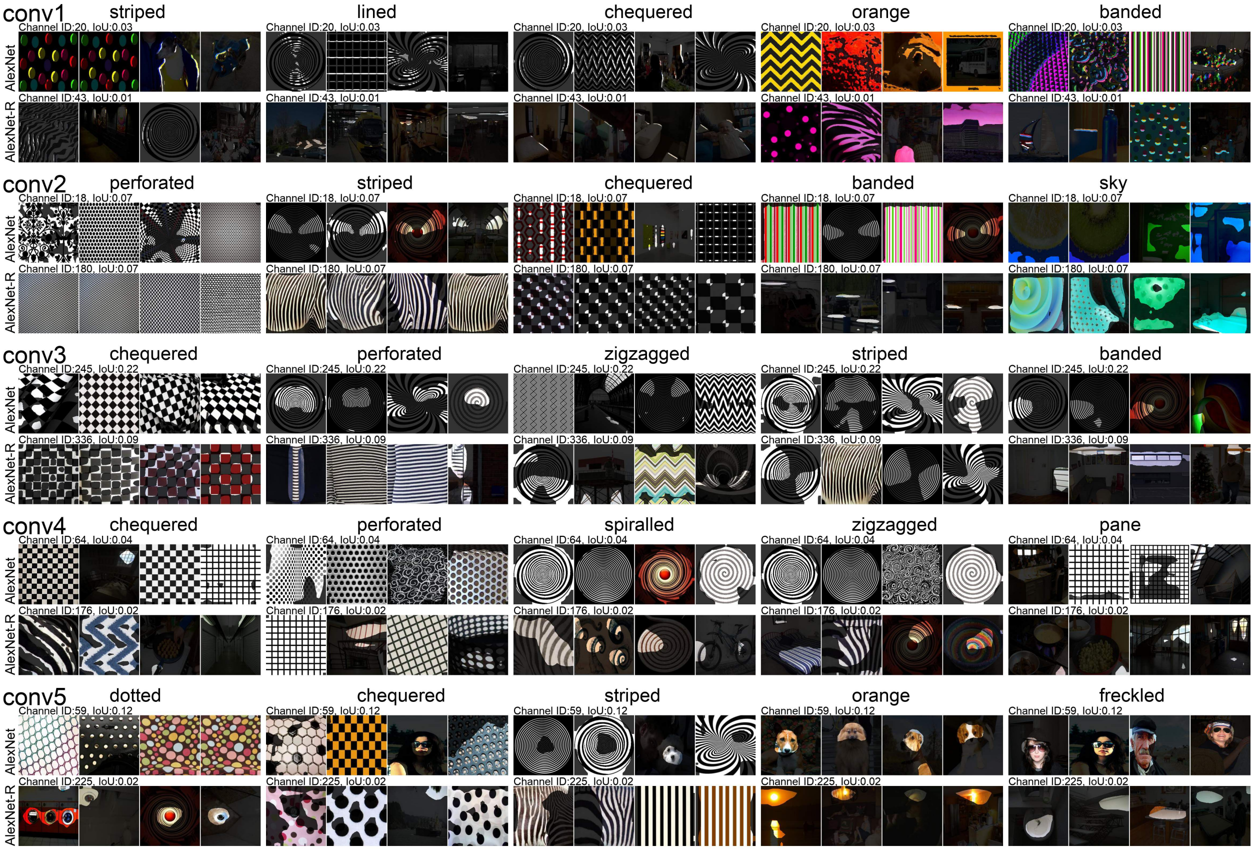
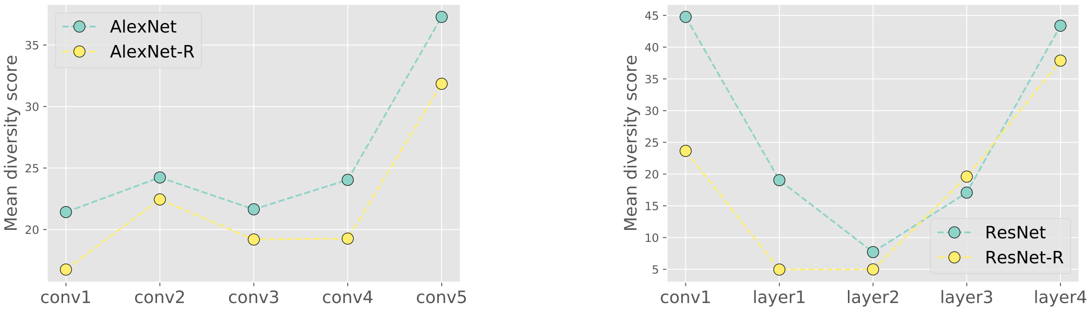


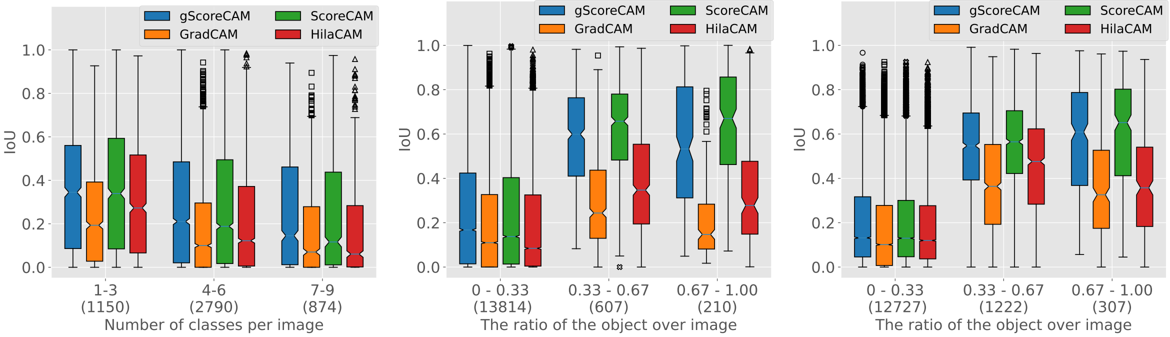
Figure 6.11: The NetDissect images preferred by the channels in the top-5 most important concepts in AlexNet (i.e. highest accuracy drop when zeroed out; see Sec. 2.6.2). For each concept, we show the highest-IoU channels.



(a) AlexNet layer-wise mean diversity

(b) ResNet layer-wise mean diversity

Figure 6.12: In each plot, we show the mean diversity scores across all channels in each layer. Both AlexNet-R and ResNet-R consistently have channels with lower diversity scores (i.e. detecting fewer unique concepts) than the standard counterparts.

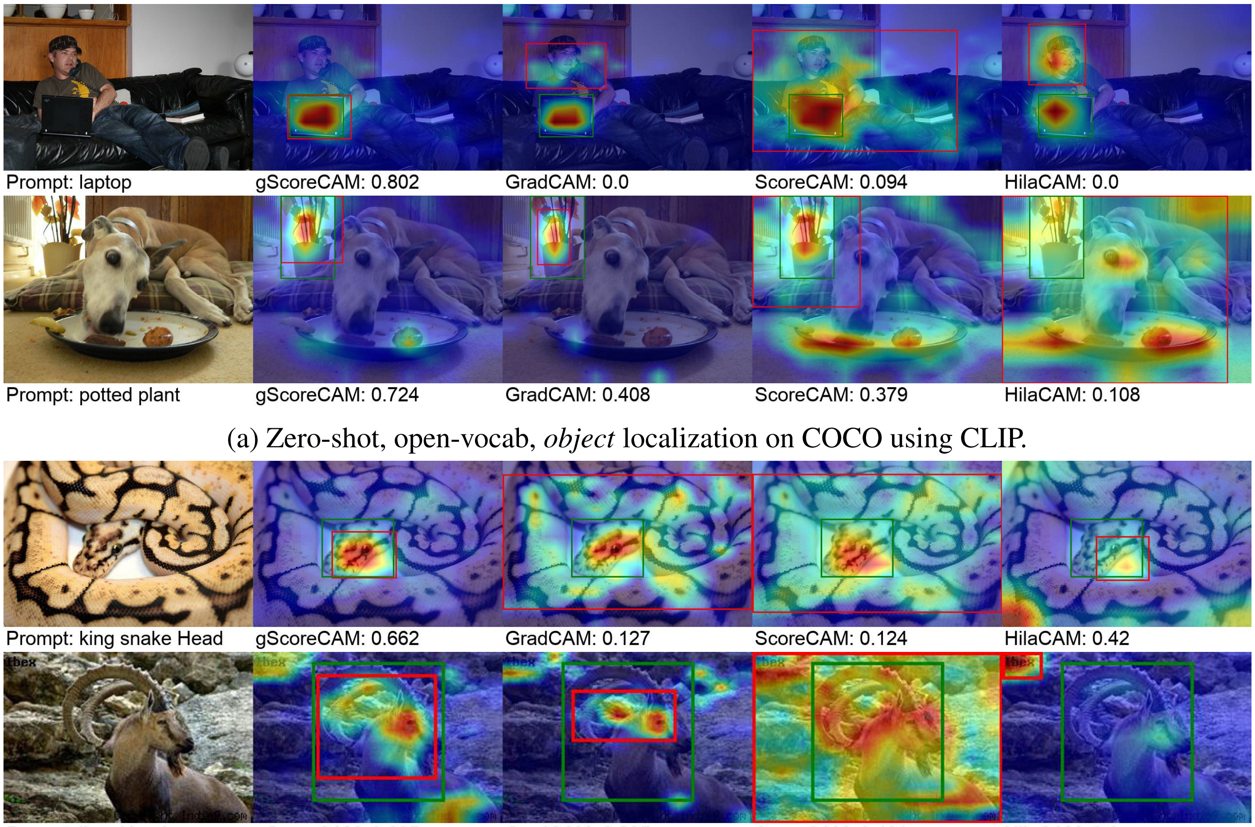


(a) On experiment of COCO multi-object, gScoreCAM has the least IoU drops as the number of classes per image increases.

(b) On COCO dataset, gScoreCAM performs the best in terms of IoU in small objects. And ScoreCAM is the best when the object size is large.

(c) On PartImageNet, results are similar to COCO. Where gScoreCAM performs better on small parts and ScoreCAM is better on larger parts.

Figure 6.13: Controlled experiments on COCO and PartImageNet. Figure 6.13a shows how IoU changes with different methods when the number of classes per image is different. Figures 6.13b and 6.13c show how the object ratio affects the methods' IoU. The number in parenthesis of x-axis on each plot is the number of samples in that group. In sum, **gScoreCAM is more accurate than other methods when a scene contains more objects (a) and object size (measured as the ratio between the object size and the image size) is smaller (b–c)**.



(a) Zero-shot, open-vocab, *object* localization on COCO using CLIP.

(b) Zero-shot, open-vocab, *part* localization on PartImageNet [39] using CLIP.

Figure 6.14: In complex scenes (i.e. not ImageNet-v2), gScoreCAM outperforms other methods, yielding more precise localization and cleaner heatmaps. IoU scores between the groundtruth (🟩) and inferred box (🟥) are shown next to each method name. More examples in Figures 6.16 and 6.17.

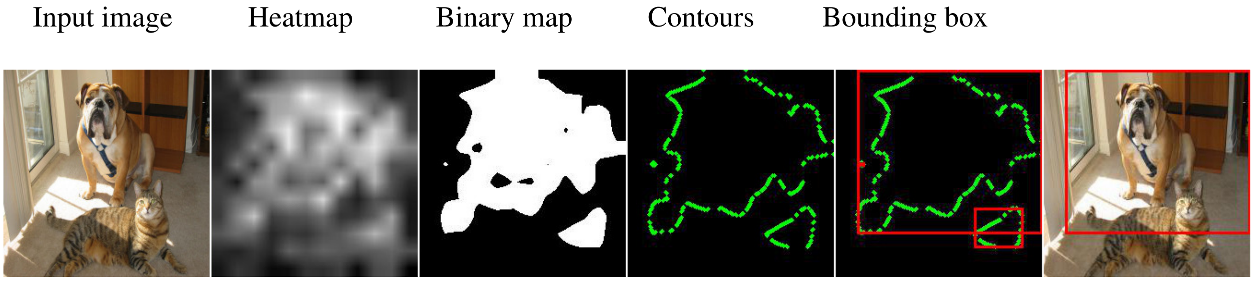


Figure 6.15: From left to right is the procedure we derive bounding box from heatmap. We first get the heatmap from input image. Then use Otsu's method to find the binary map. We find sets of bounding boxes from the contours and then choose the largest one as our final result.

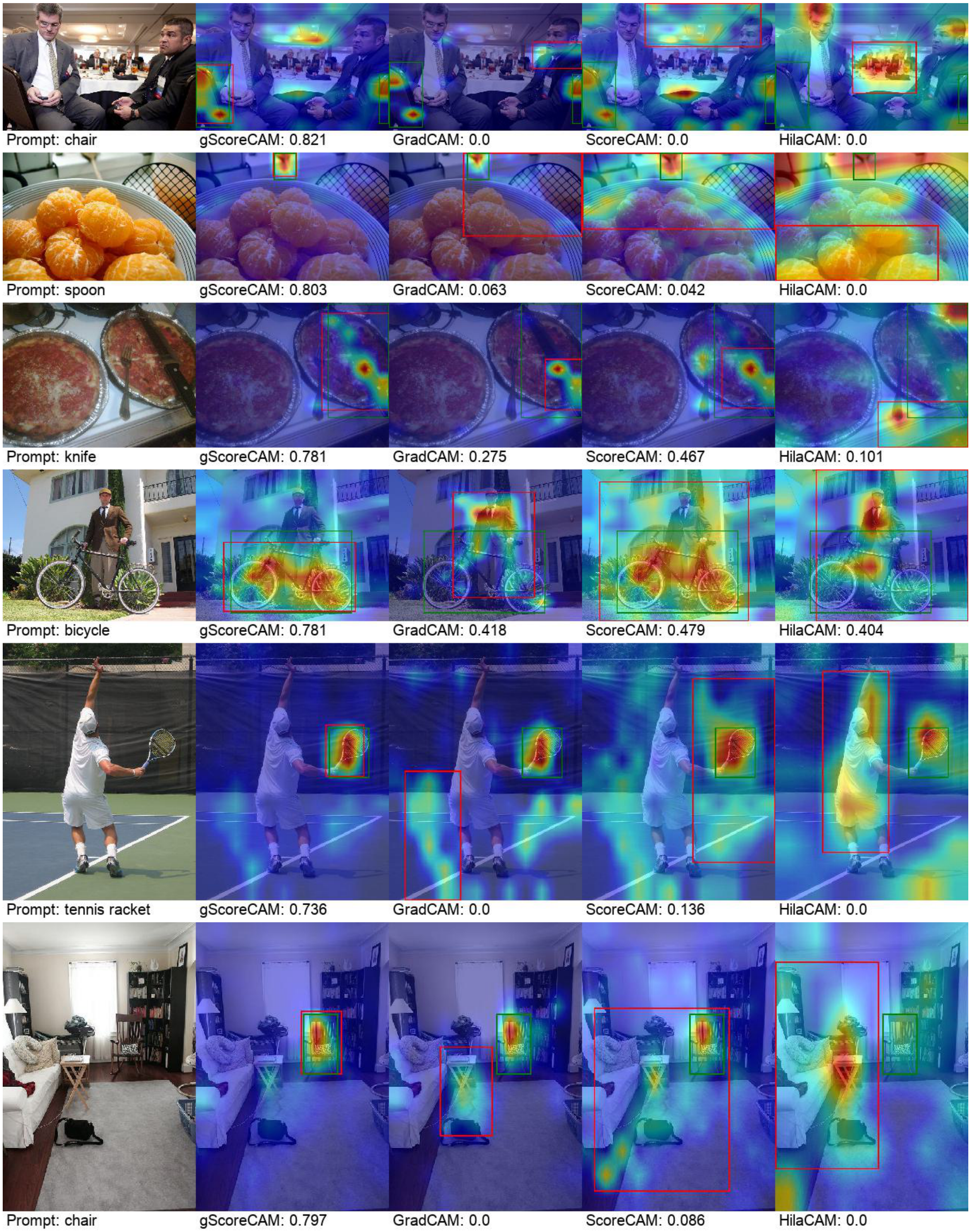


Figure 6.16: WSL results COCO dataset.

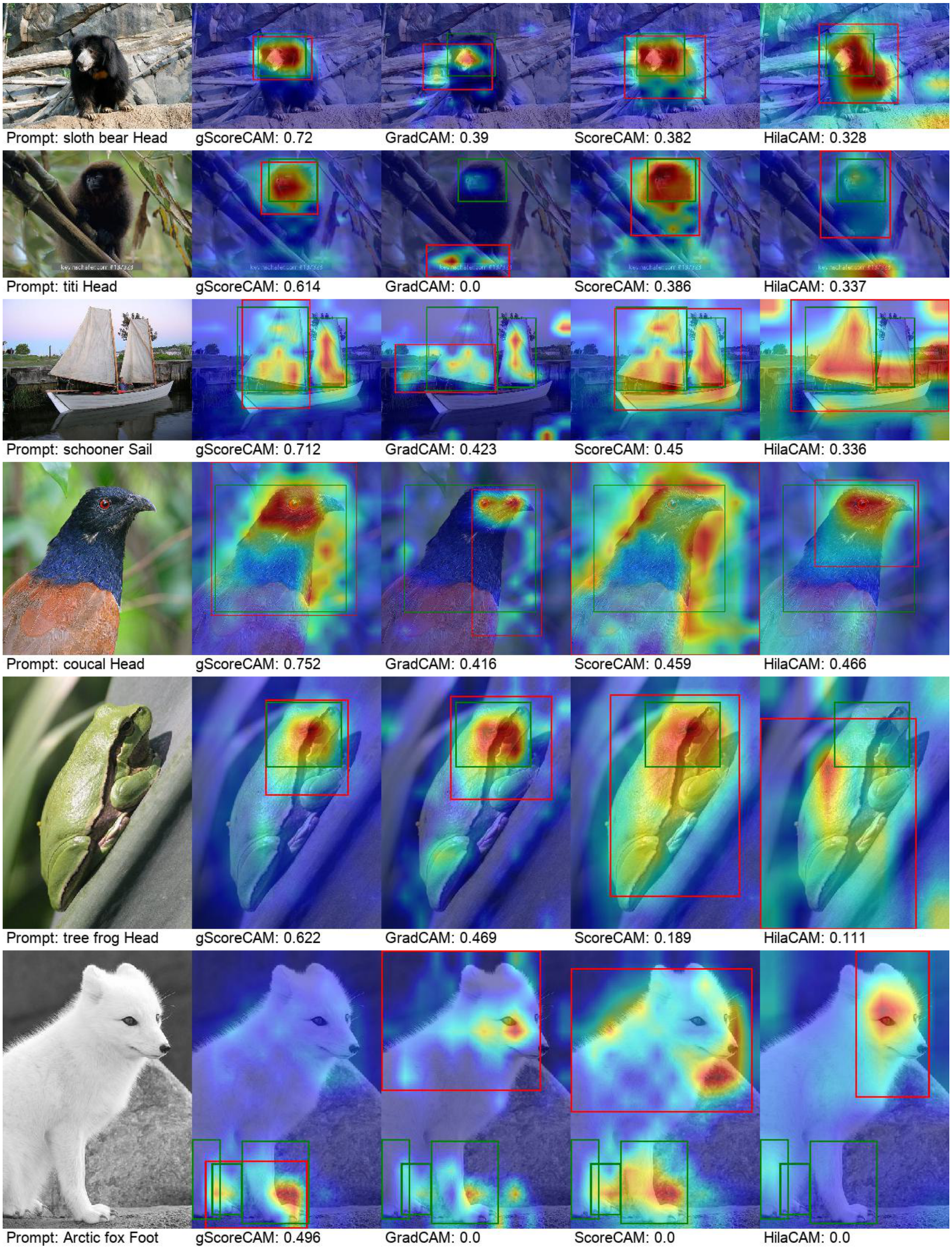
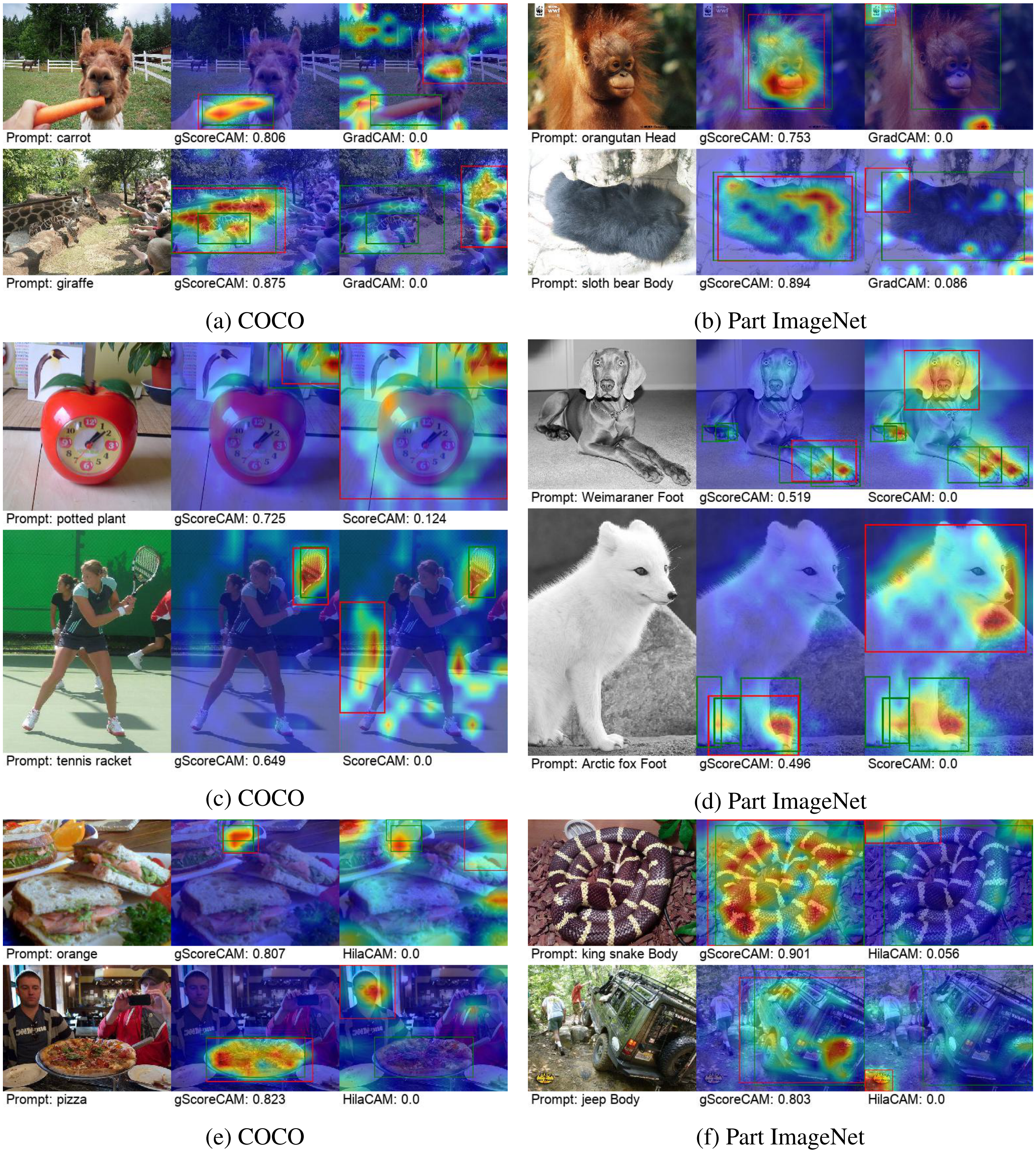


Figure 6.17: WSL results on PartImageNet dataset.



(a) COCO

(b) Part ImageNet

(c) COCO

(d) Part ImageNet

(e) COCO

(f) Part ImageNet

Figure 6.18: Sample visualize comparison between gScoreCAM to GradCAM, ScoreCAM, and HilaCAM. We can see that GradCAM always has low coverage which ScoreCAM tends to have large coverage. HilaCAM is something in the middle but have some "corner" issues. gScoreCAM can capture the target object most of the time although the resulting bounding box (red) may not be very accurate.

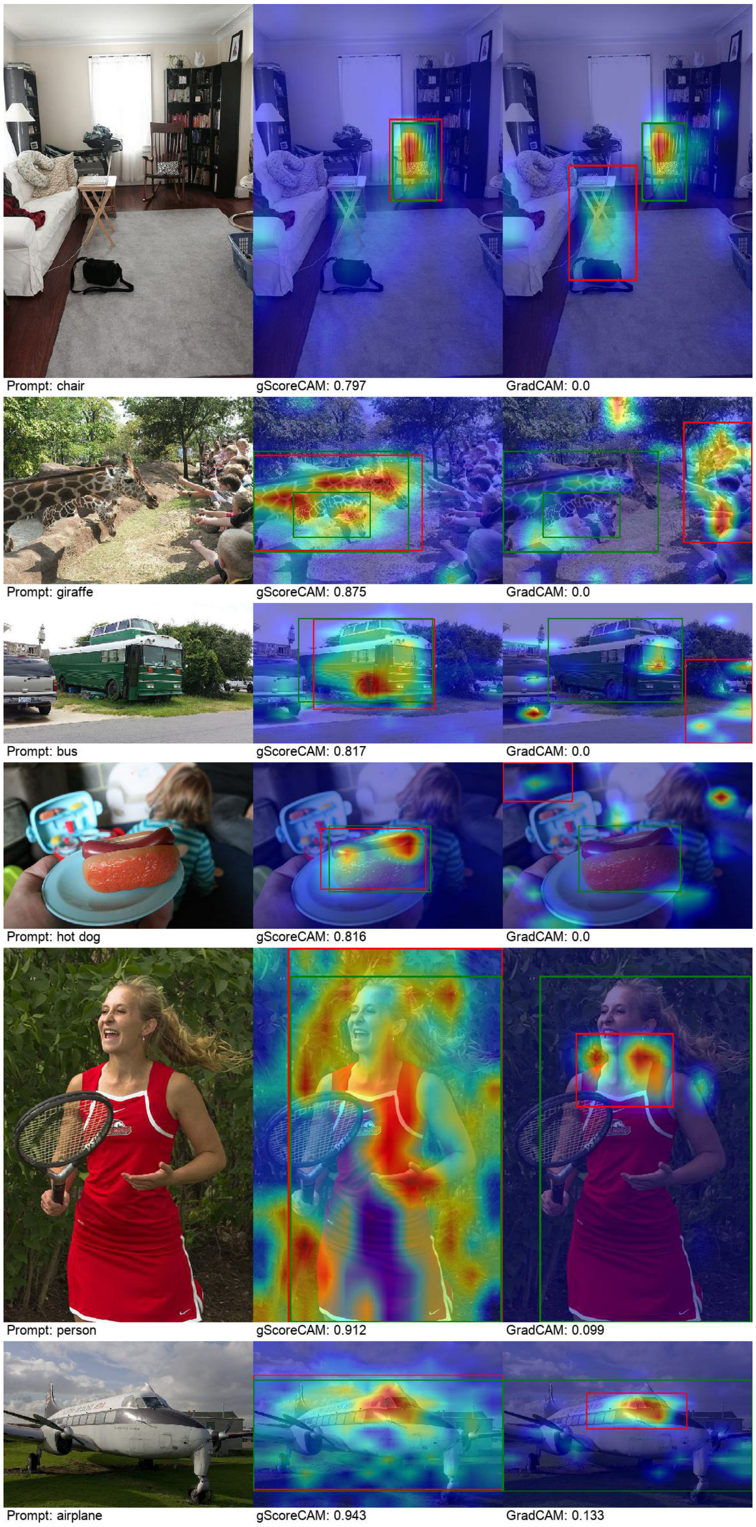


Figure 6.19: Some samples that gScoreCAM performs better than GradCAM on COCO dataset.

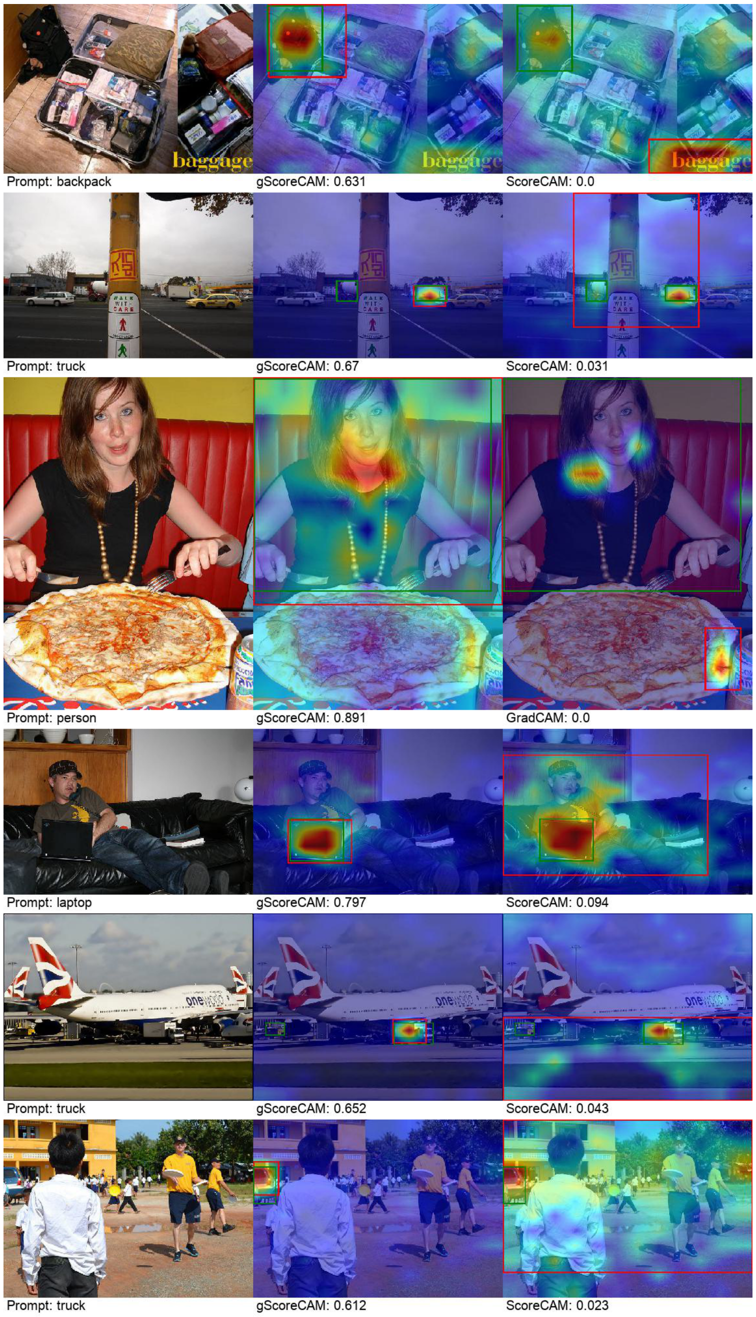


Figure 6.20: Some samples that gScoreCAM performs better than ScoreCAM on COCO dataset.

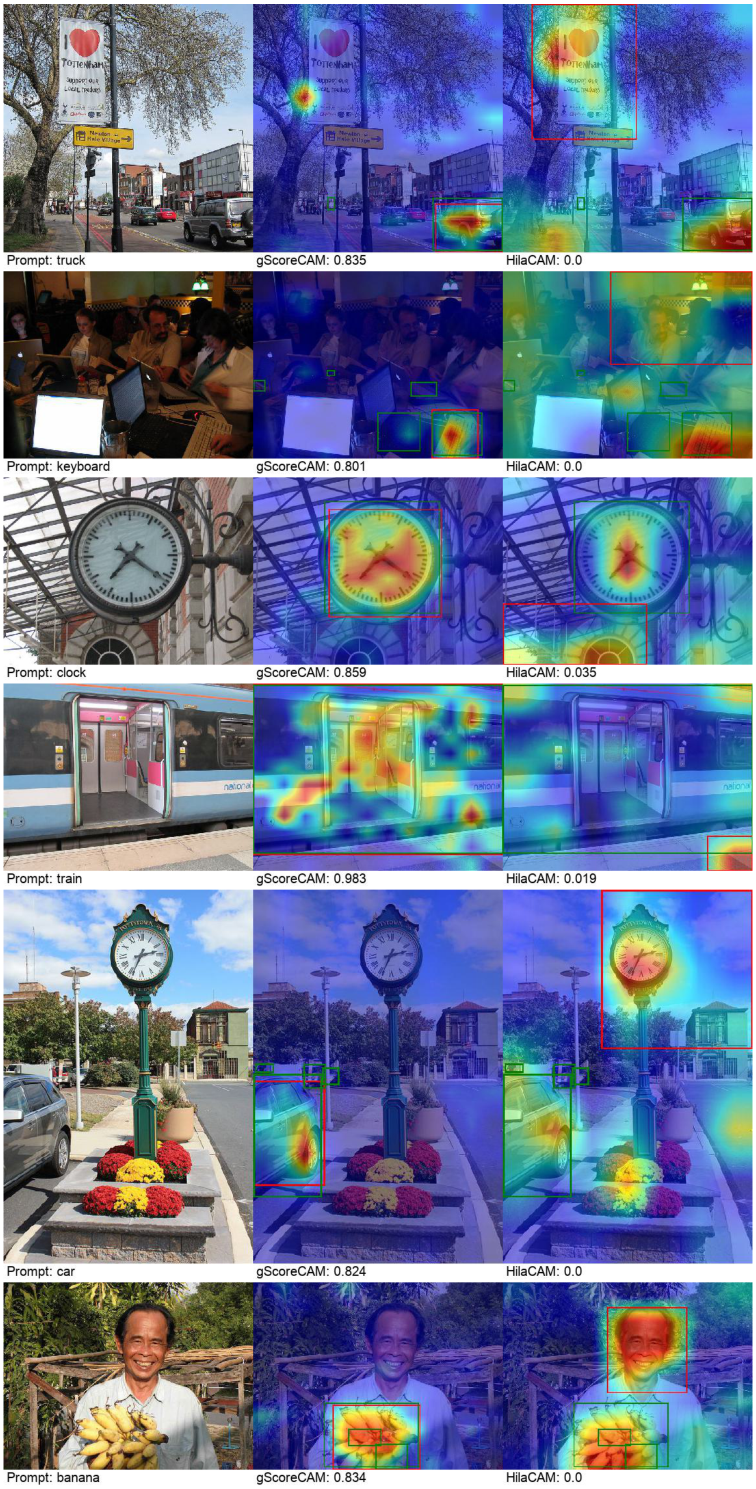


Figure 6.21: Some samples that gScoreCAM performs better than HilaCAM on COCO dataset.

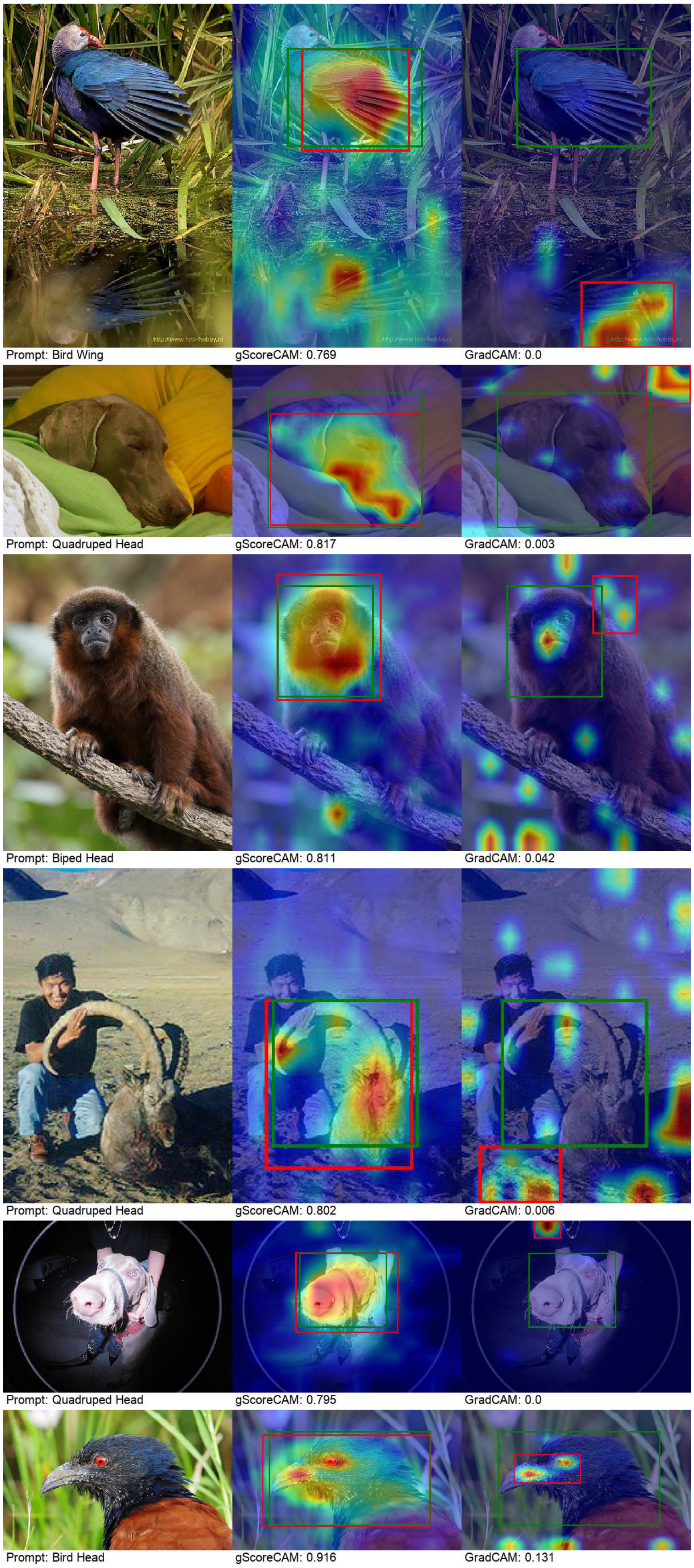


Figure 6.22: Some samples that gScoreCAM performs better than GradCAM on Part ImageNet dataset.

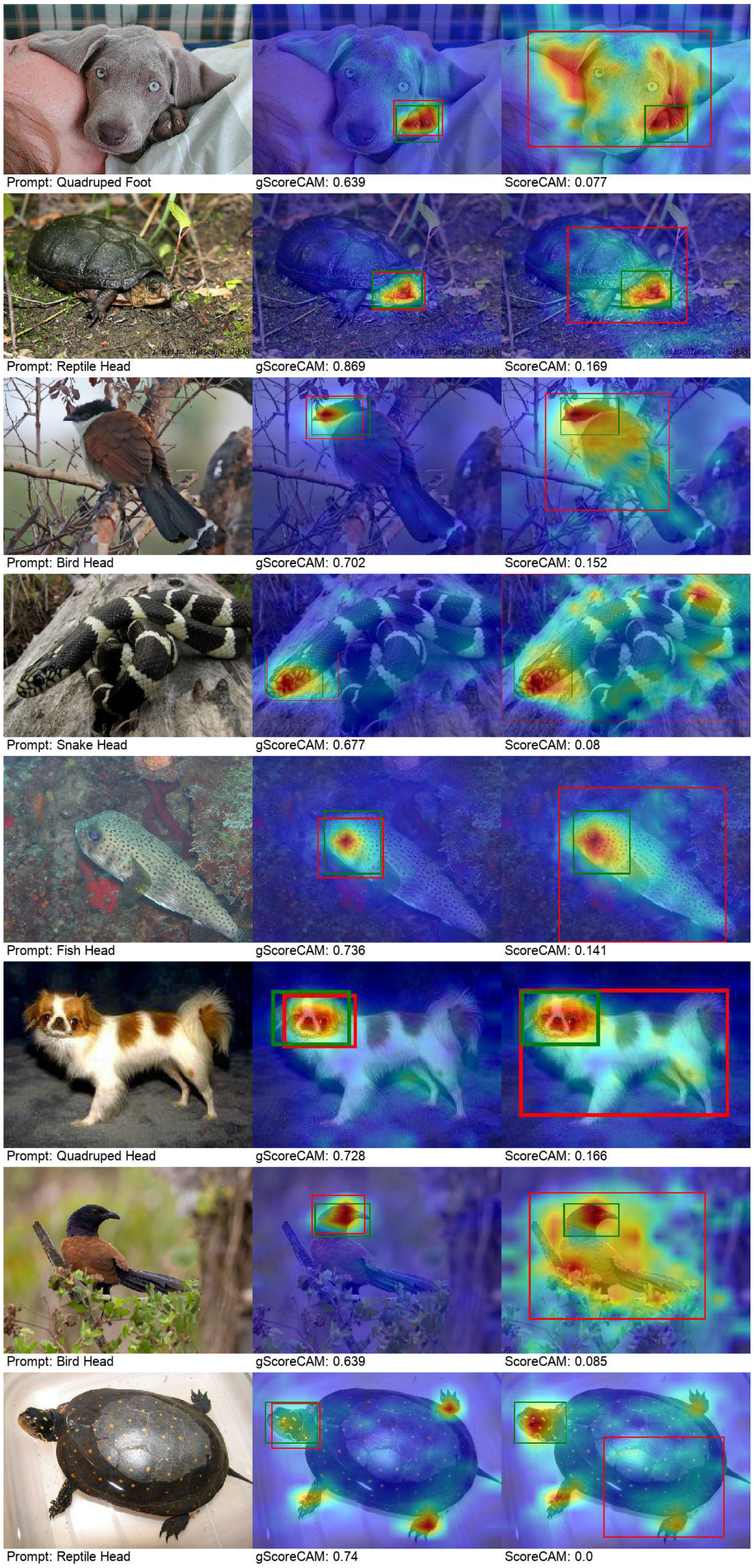


Figure 6.23: Some samples that gScoreCAM performs better than ScoreCAM on Part ImageNet dataset.

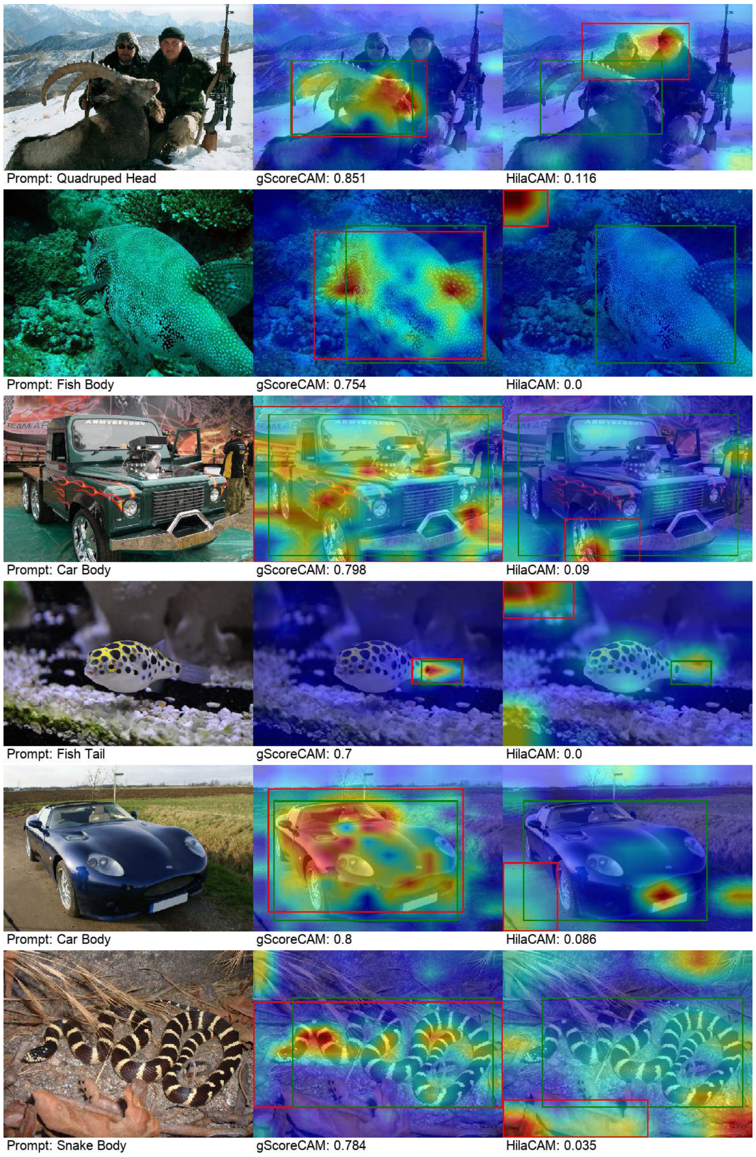


Figure 6.24: Some samples that gScoreCAM performs better than HilaCAM on Part ImageNet dataset.

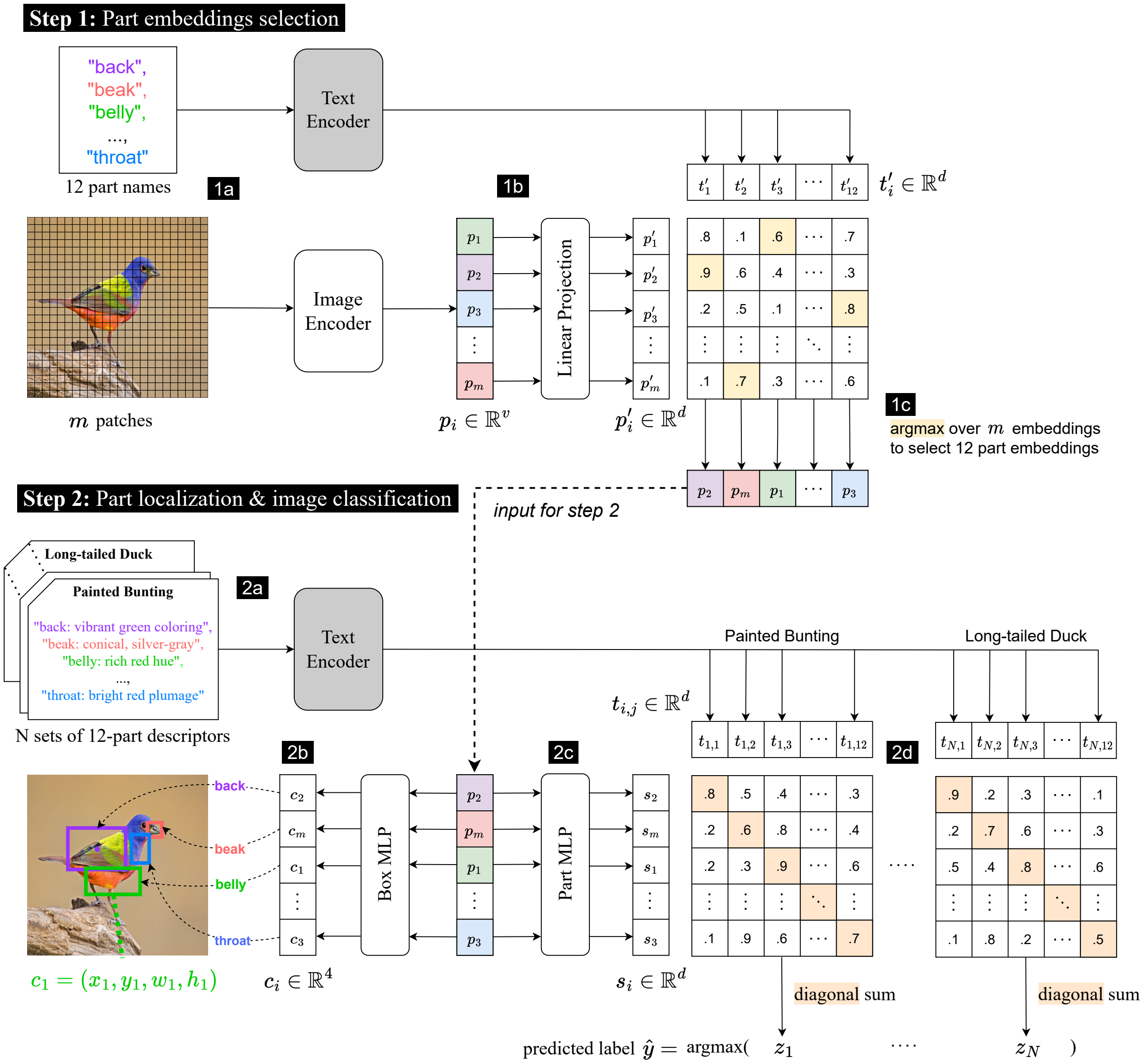

Step 1: Part embeddings selection
"back",
"beak",
"belly",
...,
"throat"
12 part names
1a
Text Encoder
1b
t'_1 t'_2 t'_3 $\cdots$ t'_{12}
$t'_i \in \mathbb{R}^d$
m patches
Image Encoder
p_1 p_2 p_3 $\vdots$ p_m
Linear Projection
p'_1 p'_2 p'_3 $\vdots$ p'_m
$p_i \in \mathbb{R}^v$
$p'_i \in \mathbb{R}^d$
.8 .1 .6 ⋯ .7
.9 .6 .4 ⋯ .3
.2 .5 .1 ⋯ .8
.1 .7 .3 ⋯ .6
1c
argmax over m embeddings
to select 12 part embeddings
p_2 p_m p_1 $\cdots$ p_3
input for step 2
Step 2: Part localization & image classification
Long-tailed Duck
Painted Bunting
"back: vibrant green coloring",
"beak: conical, silver-gray",
"belly: rich red hue",
...,
"throat: bright red plumage"
N sets of 12-part descriptors
2a
Text Encoder
Painted Bunting
Long-tailed Duck
$t_{i,j} \in \mathbb{R}^d$
$t_{1,1}$ $t_{1,2}$ $t_{1,3}$ $\cdots$ $t_{1,12}$
$t_{N,1}$ $t_{N,2}$ $t_{N,3}$ $\cdots$ $t_{N,12}$
2b
2c
2d
back
beak
belly
throat
c_2 c_m c_1 $\vdots$ c_3
Box MLP
p_2 p_m p_1 $\vdots$ p_3
Part MLP
s_2 s_m s_1 $\vdots$ s_3
.8 .5 .4 ⋯ .3
.2 .6 .8 ⋯ .4
.2 .3 .9 ⋯ .6
.1 .9 .6 ⋯ .7
.9 .2 .3 ⋯ .1
.2 .7 .6 ⋯ .3
.5 .4 .8 ⋯ .6
.1 .8 .2 ⋯ .5
$c_1 = (x_1, y_1, w_1, h_1)$
$c_i \in \mathbb{R}^4$
$s_i \in \mathbb{R}^d$
diagonal sum
diagonal sum
predicted label $\hat{y} =$ argmax(z_1 .... z_N)

Figure 6.25: During the test time using PEEB, we perform 2 steps.
Step 1: (a) Encode an input image and texts (i.e. 12 part names) by the image and text encoder to get patch embeddings p_i and text embeddings t'_i. (b) Feed p_i to Linear Projection to get p'_i in the same dimensional space with t'_i and compute dot product between $\{p'_i\}$ and $\{t'_i\}$. (c) $\arg\max$ over m embeddings to select 12 part embeddings.
Step 2: (a) Encode input texts (i.e. N sets of 12-part descriptors) with the same text encoder to get t_i. (b) Feed the selected part embeddings to Box MLP to localize parts (in center format). (c) Also feed the selected part embeddings to Part MLP to get s_i in the same dimensional space with t_i (d) Compute a dot product between $\{s_i\}$ and $\{t_i\}$, then diagonal sum for each class and $\arg\max$ over logits to get predicted label $\hat{y}$.

Descriptor embedding matching

To enhance the model's flexibility, we do not use a linear layer for classification. Instead, we adopt a strategy similar to CLIP: we compute the similarity matrix of the projected visual embeddings (image embeddings after processing by the **Part MLP**) and the text embeddings. Then, we sum the corresponding similarities of each part in the class; the class with the highest score is considered the predicted class as shown in figure 6.25, step 2, 2d. This design enables our proposed method to perform arbitrary ways of classification.

Implementation details

Our experiments are conducted under PyTorch [72]. We employ HuggingFace's [107] implementation of OWL-ViT and use their pre-trained models. The DETR losses implementation [10] is employed directly from their official implementation.

Training hyperparameters

We provide the hyperparameters of all models trained in this work. Table 6.7 shows the details of the pre-training models. Table 6.8 presents the details of the finetuned models. All trainings utilize optimizer AdamW with Plateau Scheduler.

Computational budget and infrastructures

We use 8 Nvidia RTX A100 GPUs for our experiments. The pertaining approximate takes ~24 hours on Bird-11K. The finetuning takes 2 to 4 hours with one single GPU.

Training objectives

As discussed in section 4.3.3, we have three objectives during the two training stages: (a) During the pre-training stage one, we contrastively pre-train the model to maximize the similarity between related part-descriptor pairs while minimizing the unrelated pairs using *symmetric cross-entropy (SCE)* loss [80]; (b) In pre-training stage two, we try to get rid of teacher model by mimic the teacher's choice of the boxes with SCE loss; (c) In stage two, we simultaneously train PEEB to improve box prediction with DERT losses [116].

Pre-training stage one: Symmetric cross-entropy loss for contrastive pre-training We first define the embeddings derived from the image and text encoders:

$$I'_f = \text{image\_encoder}(I) \tag{6.1}$$

where I is the input image, and $I'_f \in \mathbb{R}^{n \times d_i}$ is output image embeddings. Here, d_i is the feature dimension of the image encoder. The text embedding T_f is given by

$$T_f = \text{text\_encoder}(T) \tag{6.2}$$

where T represents the tesxt input, and $T_f \in \mathbb{R}^{m \times d_t}$. In this case, d_t is the feature dimension of the text encoder. The image embedding I'_f is then transformed by *Part MLP* layer (figure 6.25, 1b) to align its dimensions with the text embedding. This transformation is denoted as

$$I_f = \text{Part MLP}(I'_f) \tag{6.3}$$

where $I_f \in \mathbb{R}^{n \times d_t}$. The similarity matrix S between the image and text embeddings is computed as the dot product of I_f and the transpose of T_f, expressed as

$$S = I_f \cdot T_f^\top \tag{6.4}$$

where $S \in \mathbb{R}^{n \times m}$. The image logits ($S^i$) and text logits ($S^t$) are then defined as

$$S^i = \mathrm{softmax}(S, \text{axis=0}) \tag{6.5}$$

and

$$S^t = \mathrm{softmax}(S, \text{axis=1}) \tag{6.6}$$

Next, we define the symmetric cross-entropy loss for the multi-modal embeddings.

$$L_{sce} = -\frac{(\sum_i y_i^i \log(S_i^i) + \sum_m y_i^t \log(S_m^t))}{2} \tag{6.7}$$

where $y^i \in \mathbb{R}^n$ is the label for image and $y^t \in \mathbb{R}^m$ is the label for text.

Pre-training stage two: Symmetric cross-entropy loss to get rid of teacher model To get rid of the teacher model, we train PEEB to mimic the feature selection between image embedding and part name embeddings (as shown in figure 6.25, 1c). We first binary the teacher logits and consider it as the ground truth label. Then, apply the same symmetric cross-entropy loss as described in equation (6.7) with two minor differences: (1) The text input is part names rather than descriptions. (2) The *Part MLP* is replaced by *Linear Projection* (figure 6.25, 2c).

Finetuning: DETR losses DETR losses are designed to optimize the box detection performance. We employ partial losses in our training for box predictions. Specifically, we employ ℓ_1 corner-to-corner distance loss and GIoU loss. For the selected embeddings, we predict the boxes with *Box MLP* (figure 6.25, 2b)

$$B = \textit{Box MLP}(I'_f) \tag{6.8}$$

where I'_f is the image selected image embeddings from equation (6.1), $B \in \mathbb{R}^{n\times 4}$ is the predicted bounding boxes. Let $Y^{GT} \in \mathbb{R}^{n\times 4}$ be the ground truth boxes. The ℓ_1 corner-to-corner distance loss is defined as

$$L_{\ell_1} = \sum_i \left\| Y_i^{GT} - B_i \right\| \tag{6.9}$$

The GIoU loss L_{GIoU} is defined in Section 6.3.1, and the total box loss is defined as

$$L_{Box} = \frac{L_{\ell_1} + L_{GIoU}}{2} \tag{6.10}$$

Algorithm 4 Generalized Intersection over Union

```
Require: Two arbitrary convex shapes: $A, B \subseteq \mathbb{S} \in \mathbb{R}^n$
Ensure: $GIoU$
For $A$ and $B$, find the smallest enclosing convex object $C$, where $C \subseteq \mathbb{S} \in \mathbb{R}^n$
$IoU = \frac{|A \cap B|}{|A \cup B|}$
$GIoU = IoU - \frac{|C \backslash (A \cup B)|}{|C|}$
```

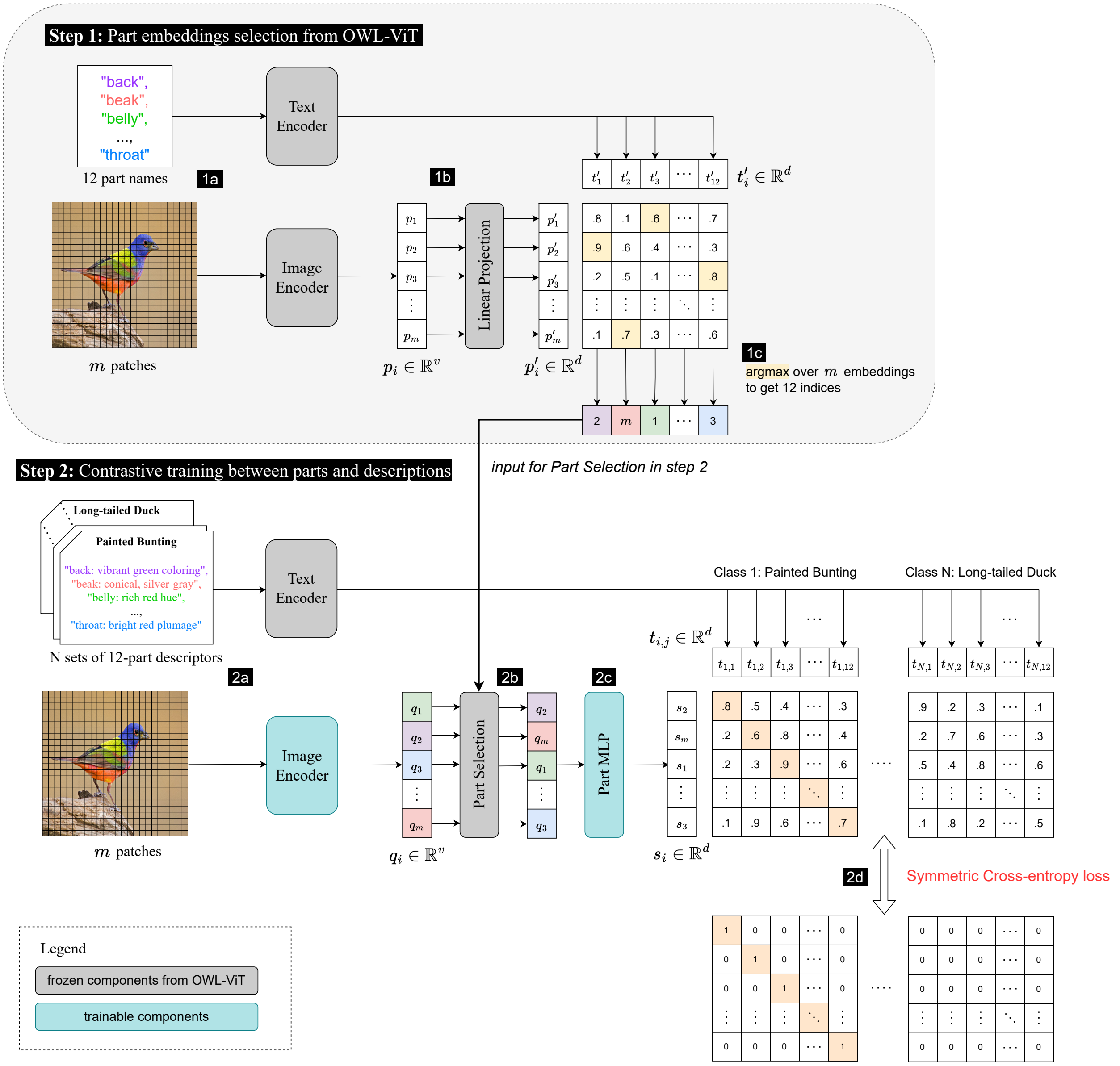


Figure 6.26: In pre-training stage 1, the objective is to let the Image Encoder learn the general representation of different parts of the birds. Therefore, in pre-training stage 1, we train the *Image Encoder* and Part MLP contrastively. During the training, the **Step 1** utilizes a teacher model (OWL-ViT$_{\text{base32}}$) to help PEEB select 12 part embeddings. In **Step 2**, we update the model with symmetric Cross-Entropy loss. Here's the flow of **Step 1**: (1a) We utilize the teacher model to encode 12 part names and the image to derive the text embedding $\boldsymbol{t}'_i$, and the patch embedding $\boldsymbol{p}_i$. (1b) Then the patch embeddings $\boldsymbol{p}$ is forwarded to Linear Projection to obtain $\boldsymbol{p}'$, matching the dimension of $\boldsymbol{t}'$. (1c) We compute the dot product between $\boldsymbol{p}$ and $\boldsymbol{t}'$ and apply $argmax$ over $\boldsymbol{p}$ to derive 12 indices. In **Step 2**: (2a), We first encode the descriptors and the image with the *Text Encoder* and *Image Encoder* to obtain descriptor embeddings $\boldsymbol{t}$ and patch embeddings $\boldsymbol{q}$. (2b), Then we select the 12 patch embeddings based on the 12 indices from (1c). (2c), The 12 patch embeddings then forwarded to Part MLP to derive $\boldsymbol{s}$, which has the same dimension as $\boldsymbol{t}$. Then, we compute the similarity matrix for the patch embedding and the descriptor embedding by computing the dot product between $\boldsymbol{s}$ and $\boldsymbol{t}$. (2d), we construct a one-hot encoded matrix based on the descriptors as the ground truth label and minimize the Symmetric Cross-Entropy loss between the similarity matrix in (2c) and the ground truth label.

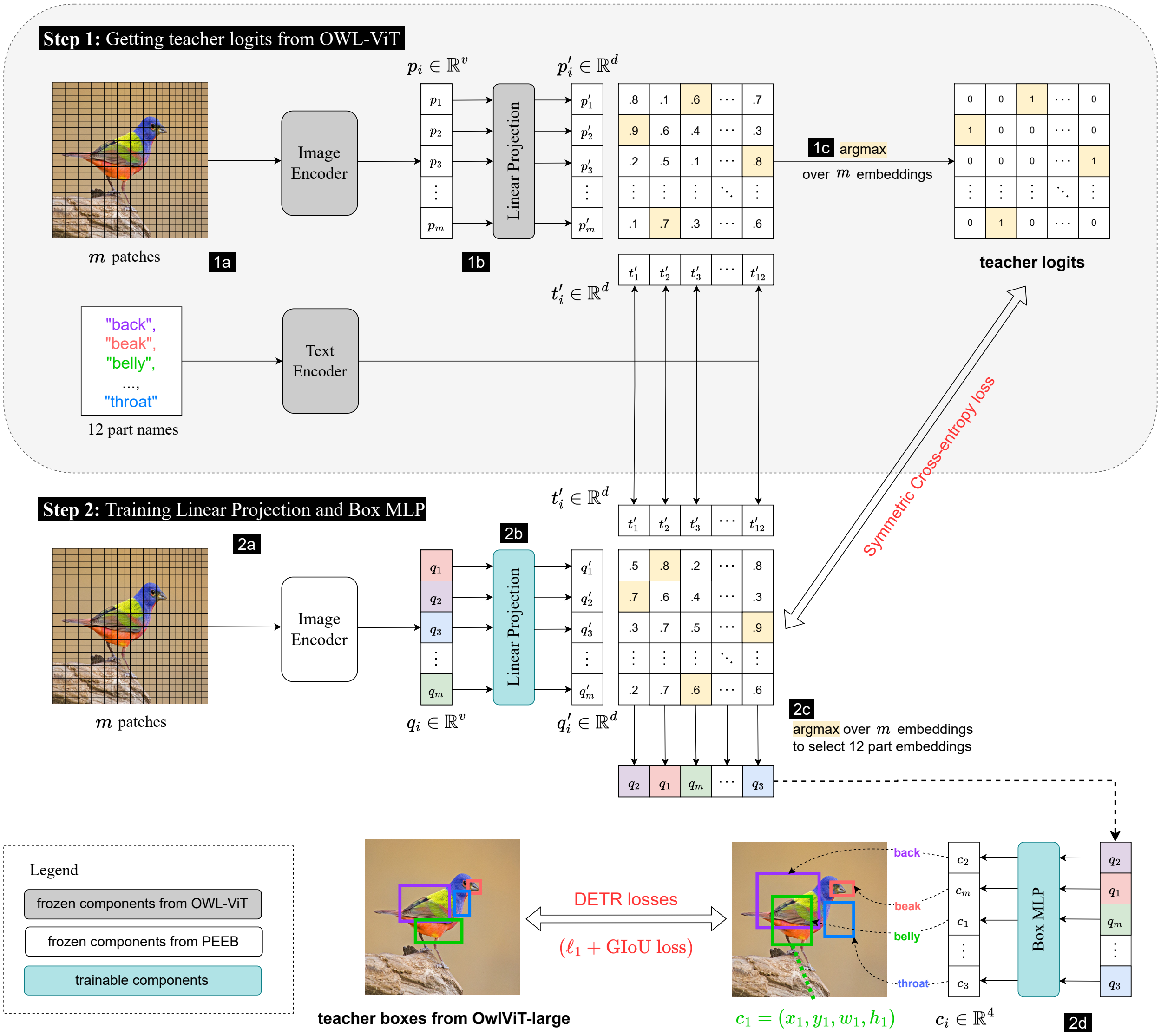


Figure 6.27: In pre-training stage 2, the goal is to eliminate the teacher model to obtain a standalone classifier. Therefore, the targeted components are Linear Projection and Box MLP. Since these two components are taking care of different functionalities for patch embedding selection and box prediction, respectively, stage 2 training is a multi-objective training. We employ Symmetric Cross-Entropy loss to learn the patch embedding selection and DETR losses to refine the box predictions. In **Step 1**: (1a), We first encode the 12 part names and the image with *Text Encoder* and *Image Encoder* to obtain the text embedding $\boldsymbol{t}'_i$ and patch embedding $\boldsymbol{p}_i$. (1b) Then the patch embeddings $\boldsymbol{p}$ is projected by Linear Projection to obtain $\boldsymbol{p}'$. (1c) We then compute dot product between $\boldsymbol{p}'$ and $\boldsymbol{t}'$ and one-hot encode the matrix via the dimension of $\boldsymbol{p}'$ to obtain the "teacher logits". In **Step 2**: (2a), We encoder the image with *Image Encoder* to obtain patch embedding $\boldsymbol{q}_i$. (2b) The patch embeddings are then being projected by Linear Projection to derive $\boldsymbol{q}'$. (2c), We compute the dot product between projected patch embeddings $\boldsymbol{q}'$ and part name embeddings $\boldsymbol{t}'$ to obtain the similarity matrix. Then, we employ Symmetric Cross-Entropy loss between the similarity matrix and the "teacher logits" derived in (1c). (2d), Meanwhile, we select the 12 part embeddings by taking argmax over $\boldsymbol{q}'$. Then, the selected part embeddings are forwarded to Box MLP to predict the coordinates of each part. We compute the DETR losses for the predicted coordinates and update the model.

Table 6.7: Pre-training details of our pre-trained models.

Model	Epoch	Batch size		LR	Weight decay	# in-batch classes		Early stop	Training set
		Train	Val			Train	Val		
Pre-training stage 1									
$\text{PEEB}_{[-\text{test}]}$	32	32	50	$2e^{-4}$	0.01	48	50	5	$\text{Bird-11K}_{[-\text{test}]}$
$\text{PEEB}_{[-\text{CUB}]}$	32	32	50	$2e^{-4}$	0.001	48	50	10	$\text{Bird-11K}_{[-\text{CUB}]}$
$\text{PEEB}_{[-\text{NAB}]}$	32	32	50	$2e^{-4}$	0.001	48	50	10	$\text{Bird-11K}_{[-\text{NAB}]}$
Pre-training stage 2									
$\text{PEEB}_{[-\text{test}]}$	32	32	50	$2e^{-5}$	0.01	48	50	5	$\text{Bird-11K}_{[-\text{test}]}$
$\text{PEEB}_{[-\text{CUB}]}$	32	32	50	$2e^{-5}$	0.001	48	50	5	$\text{Bird-11K}_{[-\text{CUB}]}$
$\text{PEEB}_{[-\text{NAB}]}$	32	32	50	$2e^{-5}$	0.001	48	50	5	$\text{Bird-11K}_{[-\text{NAB}]}$

Table 6.8: Details of our finetuned models.

Model	Fine-tune from	Epoch	Batch size	LR	Weight decay	Early stop	Training set
$\text{PEEB}^{\text{CUB}}_{[-\text{test}]}$	$\text{PEEB}_{[-\text{test}]}$	30	32	$2e^{-5}$	0.001	5	CUB
$\text{PEEB}^{\text{Akata}}_{[-\text{cub}]}$	$\text{PEEB}_{[-\text{CUB}]}$	5	32	$2e^{-5}$	0.001	5	CUB ZSL [2015]
$\text{PEEB}^{\text{SCS}}_{[-\text{cub}]}$	$\text{PEEB}_{[-\text{CUB}]}$	5	32	$2e^{-5}$	0.001	5	CUB-SCS
$\text{PEEB}^{\text{SCE}}_{[-\text{cub}]}$	$\text{PEEB}_{[-\text{CUB}]}$	5	32	$2e^{-5}$	0.001	5	CUB-SCE
$\text{PEEB}^{\text{SCS}}_{[-\text{nab}]}$	$\text{PEEB}_{[-\text{NAB}]}$	5	32	$2e^{-5}$	0.001	5	NABirds-SCS
$\text{PEEB}^{\text{SCE}}_{[-\text{nab}]}$	$\text{PEEB}_{[-\text{NAB}]}$	5	32	$2e^{-5}$	0.001	5	NABirds-SCE

6.3.2 Model and dataset notations

Dataset notations

Following the conventional setup of ZSL, we execute certain exclusions to make sure none of the test classes or descriptors are exposed during pre-training. That is, Bird-11K$_{[-\mathsf{CUB}]}$ and Bird-11K$_{[-\mathsf{NAB}]}$ exclude all CUB and NABirds classes, respectively. For GZSL, we exclude all test sets in CUB, NABirds, and iNaturalist, denoted as Bird-11K$_{[-\mathsf{test}]}$. We provide detailed statistics for the three pre-training sets in Table 6.9.

Table 6.9: Three pre-training splits for PEEB.

Training set	Number of images		Number of classes	
	Train	Val	Train	Val
Bird-11K$_{[-\mathsf{test}]}$	234,693	29,234	10,740	9,746
Bird-11K$_{[-\mathsf{CUB}]}$	244,182	28,824	10,602	9,608
Bird-11K$_{[-\mathsf{NAB}]}$	216,588	27,996	10,326	9,332

Model notations

We adopt a strategy based on the datasets excluded during training to simplify our model naming convention. Specifically:

- PEEB$_{[-\mathsf{test}]}$ is pre-trained model using Bird-11K$_{[-\mathsf{test}]}$ datset.
- PEEB$_{[-\mathsf{CUB}]}$ is pre-trained model using the Bird-11K$_{[-\mathsf{CUB}]}$ dataset.
- PEEB$_{[-\mathsf{NAB}]}$ is pre-trained model using the Bird-11K$_{[-\mathsf{NAB}]}$ dataset.

We named finetuned models after the pre-trained model and the finetuned training set. For example, PEEB$^{\mathsf{CUB}}_{[-\mathsf{test}]}$ is finetuned from PEEB$_{[-\mathsf{test}]}$, on CUB training set.

6.3.3 Generating part-based descriptors

CUB annotations initially comprise 15 bird parts. However, distinctions between the left and right part are not essential to our method, we merge them into a single part (i.e., “left-wing” and “right-wing” are merged into “wings”) Hence, we distilled the original setup into 12 definitive parts: *back, beak, belly, breast, crown, forehead, eyes, legs, wings, nape, tail, throat*. To compile visual part-based descriptors for all bird species within Bird-11K, we prompted GPT-4 [70] with the following input template:

```
A bird has 12 parts:  back, beak, belly, breast, crown, forehead, eyes,
legs, wings, nape, tail and throat.  Visually describe all parts of
{class name} bird in a short phrase in bullet points using the format
'part:  short phrase'
```

Where `{class name}` is substituted for a given bird name (e.g., PaintedBunting). The output is a set of twelve descriptors corresponding to twelve parts of the query species. e.g. The response for `Cardinal` is:

```
Cardinal: {
    back: vibrant red feathers,
    beak: stout, conical, and orange,
    belly: light red to grayish-white,
    breast: bright red plumage,
    crown: distinctive red crest,
    forehead: vibrant red feathers,
    eyes: small, black, and alert,
    legs: slender, grayish-brown,
    wings: red with black and white accents,
    nape: red feather transition to grayish-white,
    tail: long, red, and wedge-shaped,
    throat: bright red with sharp delineation from white belly
}
```

6.3.4 Datasets

🦜 Bird-11K

We provide a brief statistic of Bird-11K in table 6.10. Bird-11K is a diverse and long-tailed avian dataset that only includes bird images. The descriptors generated by GTP4 are in English and only describe the visual features of the corresponding class. We propose Bird-11K for academic research only.

Table 6.10: Number of images and species of different bird datasets. Our proposed dataset Bird-11K includes almost all avians on Earth.

Dataset	# of **Images**	# of **Species**
CUB-200-2011 [104]	12,000	200
Indian Birds [99]	37,000	25
NABirds v1 [100]	48,000	400
Birdsnap v7 [6]	49,829	500
iNaturalist 2021-birds [101]	74,300	1,464
ImageNet-birds [21]	76,700	59
BIRDS 525 [78]	89,885	525
Macaulay Library at the Cornell Lab of Ornithology	55,283	10,534
Bird-11K (Raw Data)	440,934	11,097
Bird-11K (pre-training set)	294,528	10,811

Data splits We provide data splits and metadata, e.g., file names, image size, and bounding boxes, along with the instruction of Bird-11K construction in our repository. Note that the Bird-11K dataset is for pre-training purposes; it is important to execute exclusion based on the test set.

License and terms

- CUB [104]: The dataset can be freely used for academic and research purposes; commercial use is restricted.

- Indian Birds [99]: CC0: Public Domain.

- NABirds v1 [100]: For non-commercial research purposes, other use is restricted [1] here for detail: .

- Birdsnap v7 [6]: The dataset creator provides no specific license or terms of use. We only use this dataset for academic research until more specific details can be obtained.

- iNaturalist 2021-birds [101]: CC0: Public Domain.

- ImageNet-birds [21]: BSD-3-Clause license.

- BIRDS 525 [78]: CC0: Public Domain

- Cornell eBird: We used the following 55,384 recordings from the Macaulay Library at the Cornell Lab of Ornithology. The data is for academic and research purposes only, not publicly accessible unless requested. (Please refer to our Supplementary Material for the full list): ML187387391, ML187387411, ML187387421, ML187387431, ML262407521, ML262407481, ML262407531, ML262407491, ML262407511, ML257194111 ML257194071, ML257194081, ML257194061, ML495670791, ML495670781, ML495670801, ML495670771, ML183436431, ML183436451, ML183436441 ML183436411, ML183436421, ML256545901, ML256545891, ML256545841, ML256545851, ML256545831, ML169637941, ML238083081, ML169637881 ML169637911, ML238083111, ML238083051, ML169637971, ML299670841, ML64989231, ML299670831, ML64989241, ML299670791, ML64989251 ML246866001, ML246865941, ML246866011, ML246865961, ML246865971, ML333411961, ML240835531, ML240835541, ML240835701, ML240835591 ML245260391, ML245260341, ML245260371, ML245260411, ML245260421, ML245260431, ML245260441, ML240866351, ML240866331, ML240866321 ML240866341, ML240866371, ML248318661, ML248318571, ML248318591, ML248318581, ML248318631, ML245204281, ML245204311, ML245204371 ML245204381, ML245204291, ML245603571, ML245603521, ML245603511, ML245603491, ML245603501, ML245603601, ML245257771, ML245257651 ML245257631, ML245257661, ML245257761, ML247221051, ML247221061, ML247221071, ML247221081, ML240365811, ML240365751, ML240365781 ML240365761, ML300579541, ML247298551, ML247298541, ML247298561, ML247298611, ML247298571, ML247298591, ML247298601, ML247298631...

🦮 Dog-140

To pre-train PEEB on dogs, we construct Dog-140 by combining dog images from ImageNet-21K and Stanford Dogs. Specifically, we selected 189 dog classes from ImageNet-21K, and based on Fédération Cynologique Internationale (FCI) [29], we merged them with 120 classes from Stanford Dogs, ending up with 140 classes. After merging, Dog-140 has 206,076 images in total. We provide a class distribution analysis in Figure 6.28, where we can find that Dog-140 is roughly class-balanced.

[1] See Terms of Use

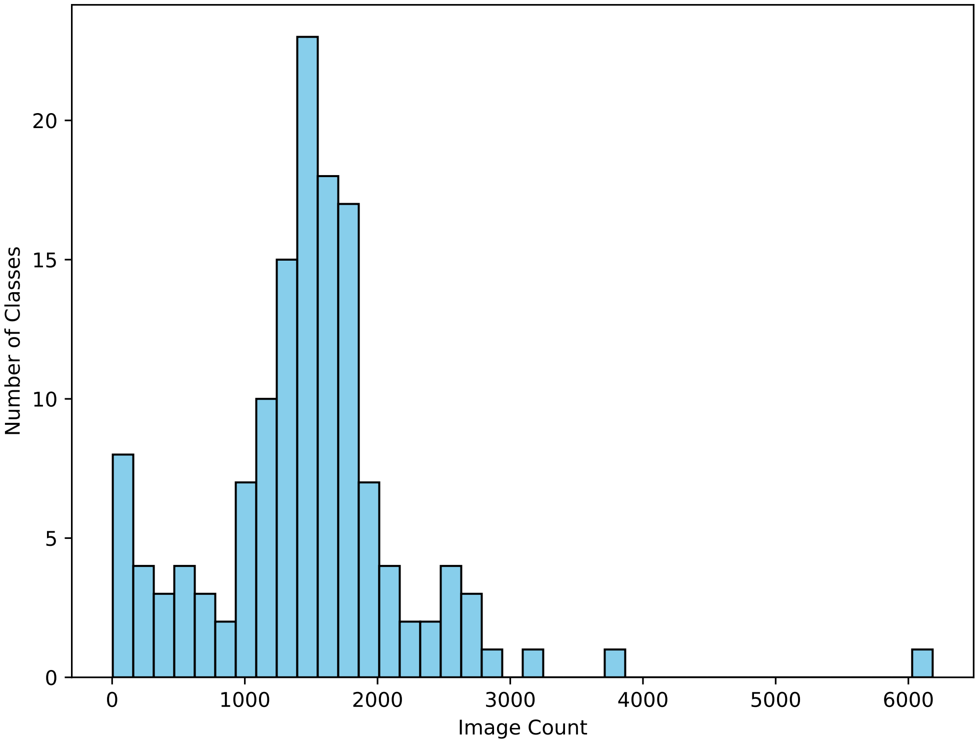


Figure 6.28: The class distribution of Dog-140 dataset. The histogram indicates that most classes in Dog-140 have around 1,000 to 2,000 images.

Data splits Similar to Bird-11K, we provide data splits and metadata, e.g., file names, image size, and bounding boxes, along with the instruction of Dog-140 construction in our repository.

License and terms

- Stanford Dogs [47]: The dataset was constructed using images and annotations from ImageNet. Therefore, all the images (including those presented in the paper) follow the ImageNet license.
- ImageNet-21K [21]: BSD-3-Clause license, non-commercial.

6.3.5 Additional results

PEEB outperforms M&V in CUB and NABirds in ZSL setting

To rigorously evaluate the ZSL capabilities of our pre-trained models, we introduce a stress test on the CUB and NABirds datasets. The crux of this test involves excluding all classes from the target dataset (CUB or NABirds) during the pre-training. The exclusion ensures that the model has no prior exposure to these classes. Subsequently, we measure the classification accuracy on the target dataset, comparing our results against benchmarks set by CLIP and M&V in the scientific name test. In this experiment, we consider the scientific name test a ZSL test for CLIP and use them as the baseline because the frequencies of scientific names are much lower than common ones.

Experiment To conduct this test, we pre-train our model on Bird-11K$_{[-\mathsf{CUB}]}$ and Bird-11K$_{[-\mathsf{NAB}]}$, which deliberately exclude images bearing the same class label as the target dataset. Specifically, we test on our pre-train model PEEB$_{[-\mathsf{CUB}]}$ and PEEB$_{[-\mathsf{NAB}]}$ (see Table 6.7 for details), respectively.

Results The primary objective is to ascertain the superiority of our pre-trained model, PEEB, against benchmarks like CLIP and M&V. For CUB, our method reported a classification accuracy of 17.9%, contrasting the 5.95% and 7.66% achieved by CLIP and M&V, respectively, as shown in table 6.11. The PEEB score, which is marginally higher (+10) than M&V, highlights the advantages of our method that utilizes component-based classification. On the NABirds, our method surpassed the CLIP and M&V by (+1) point. The performance disparity between CUB and NABirds can be attributed to two factors: the elevated complexity of the task (555-way classification for NABirds versus 200-way for CUB) and the marked reduction in training data. An auxiliary observation, detailed in Section 6.3.5, indicates that our pre-trained model necessitates at least 250k images to achieve admirable classification accuracy on CUB, but we only have 210k images training images in Bird-11K$_{[-\mathsf{NAB}]}$ (Table 6.9).

Table 6.11: Stress test results on CUB and NABirds datasets. Despite the ZSL challenge, our method consistently surpasses CLIP and M&V. This underscores the robust generalization of our approach, which leverages descriptors for classification.

Method	CLIP	M&V	PEEB (ours)
CUB	5.95	7.66	**17.90**
NABirds	4.73	6.27	**7.47**

Performance measurement on different noisy levels

In our evaluations, we discerned a marked performance disparity between the iNaturalist dataset and others. Probing this further, we identified image noise as a principal contributor to these discrepancies.

Experiment A qualitative assessment of the iNaturalist test images revealed a significantly higher noise level than CUB or NABirds. To systematically study this, we utilize the object detector OWL-ViT$_{\mathsf{large}}$ to measure the size of the bird within the images. We formulated two filtered test sets based on the detector's output, categorizing them by the bird's size, specifically, the detected bounding box. Images were filtered out if the bird's size did not exceed predetermined thresholds (areas of 100^2 or 200^2 pixels). Larger birds naturally reduced other content by occupying more image space, thus serving as a proxy for reduced noise. All three test sets, including the original, were evaluated using our pre-trained model PEEB$_{[-\mathsf{test}]}$.

Results The results presented in table 6.12 reveal a clear trend: as the image noise level decreases, the classification accuracy consistently improves, with gains ranging from (+6 to +17) points across the various methods. Notably, cleaner images consistently yield better results. At each noise level, our method outperforms the alternatives. While our method exhibits an impressive (+17 points) accuracy boost on the cleanest test set, this substantial gain also indicates that our model is sensitive to image noise.

Table 6.12: The table showcases the classification accuracies on iNaturalist as we vary the noise levels. The data underscores that the performance disparity on iNaturalist is predominantly due to image noise. While all methods improve with cleaner images, our model exhibits the most substantial gains, particularly in the least noisy sets.

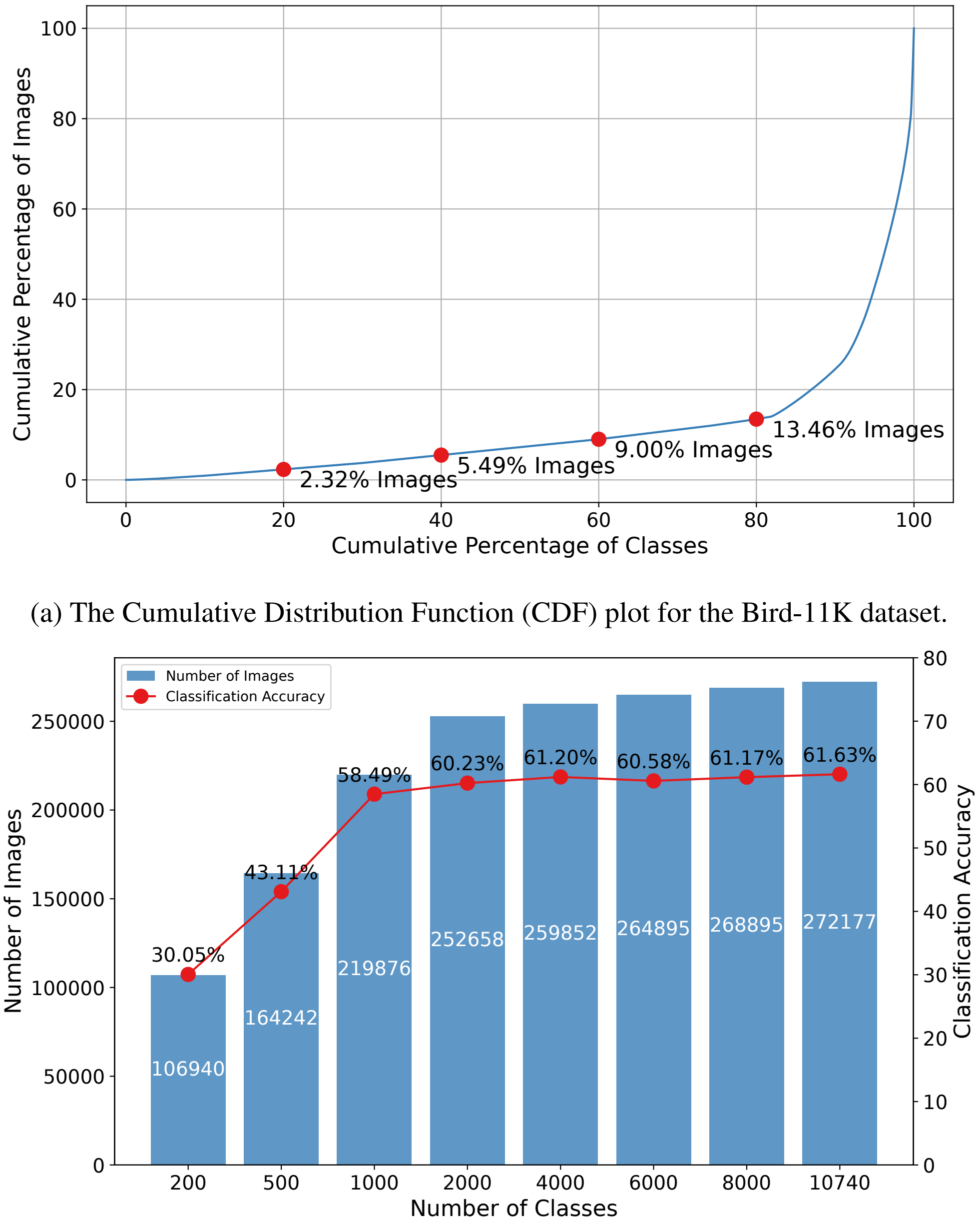
Splits	CLIP	M&V	PEEB (ours)
Original	16.36	17.57	**25.74**
> 100^2 pixels	20.18	21.66	**35.32**
> 200^2 pixels	22.88	24.90	**42.55**

Number of training images is the most critical factor towards classification accuracy

Bird-11K, as shown in figure 6.29a, is a highly imbalanced dataset characterized by a large amount of long-tailed classes. We conduct a comprehensive study to discern how variations in the number of classes and images affect the classification accuracy of our pre-trained models. Predictably, the volume of training images occurred as the most influential factor. However, a noteworthy observation was that the abundance of long-tailed data enhanced the model's accuracy by approximately +1.5 points.

Experiment We curated eight training sets based on varying class counts: 200, 500, 1,000, 2,000, 4,000, 6,000, 8,000, and 10,740. For each set, we maximized the number of training images. It is important to note that a set with a lesser class count is inherently a subset of one with a higher count. For instance, the 500-class set is a subset of the 2,000-class set. For each split, we apply the same training strategy as in Section 4.3.3, and choose the checkpoint with the best validation accuracy. We consider the CUB test set as a generic testing benchmark for all variants.

Results As illustrated in Figure figure 6.29b, there is a pronounced correlation between the increase in the number of images and the corresponding surge in accuracy. For instance, an increment from 106K to 164K images led to a rise in classification accuracy from 30.05% to 43.11%. The accuracy appears to stabilize around 60% when the image count approaches 250K. This trend strongly suggests that the volume of training images is the most critical factor for the pre-trained model. We believe that the accuracy of the pre-trained model could be further enhanced if enough data is provided. Interestingly, a substantial amount of long-tailed data bolsters the model's performance, evident from +1.5 points accuracy improvement when comparing models trained on 2,000 classes to those on 10,740 classes. Note that the additional classes in the latter set averaged merely 2.2 images per class.



(a) The Cumulative Distribution Function (CDF) plot for the Bird-11K dataset.

(b) Correlation between the number of training Figures/chapter4/classes and accuracy.

Figure 6.29: The CDF plot (a), underscores significant imbalance of the Bird-11K dataset. While the dataset has abundant long-tailed classes, e.g., a striking 80% of the classes contribute to only 13.46% of the entire image count. The plot (b) showcases the correlation between the number of training Figures/chapter4/classes and the resulting classification accuracy. As the image count grows, there is a noticeable surge in accuracy, which nearly stabilizes upon surpassing 250K images. Additionally, a significant amount of long-tailed data contributes to a +1.5 points boost in accuracy.

Ablation study on the influence of parts utilized

In this ablation study, we aimed to measure the impact of varying the number of distinct "parts" (back, beak, belly, breast, crown, forehead, eyes, legs, wings, nape, tail, and throat) used in our model. We experiment with a range from a single part to all 12 identifiable parts. Interestingly, even with a solitary part, the model could make correct predictions, though there was an evident decline in performance, approximately -20 points.

Experiment Our testing ground is the pre-trained model PEEB$_{[-\text{test}]}$, evaluated against the CUB test set. We assessed the model's prowess utilizing various subsets of parts: 1, 3, 5, 8, and all 12. These subsets were derived based on the frequency of visibility of the parts within the CUB dataset, enabling us to compare the model's performance when relying on the most frequently visible parts versus the least. For comparison, we also conduct a similar experiment on M&V, where we only use 1, 3, 5, 8, and 12 descriptors (if possible).

Results Relying solely on the most frequent part led to a decline in classification accuracy by around -20 points, registering at 45.44% (table 6.13). In contrast, utilizing the least frequent part resulted in a sharper drop of around -27, with an accuracy of 37.02%. As the model was furnished with increasing parts, its accuracy improved incrementally. The data underscores that optimal performance, an accuracy of 64.33%, is attained when all 12 parts are included. For M&V, the accuracy keeps increasing homogeneously from 5 to 12 descriptors, hinting that accuracy may increase further by increasing the number of descriptors.

Table 6.13: Classification accuracy on the CUB test set that uses a different number of parts. Performance dips significantly with just one part, especially for the least visible ones. Maximum accuracy is reached with all 12 parts. The last row of the table also shows the accuracy of [60] method which employs a different number of parts. It is evident that their method is insensitive to the number of parts used, which may not reflect a realistic scenario.

Number of Parts (descriptors)	1	3	5	8	12
Accuracy (most frequent parts)	45.44	56.48	59.89	61.32	**64.33**
Accuracy (least frequent parts)	37.02	55.51	60.04	61.13	**64.33**
Accuracy of [60]	51.93	52.87	52.83	53.33	53.92

Training is essential for PEEB's classification efficacy

In this ablation study, we highlight the pivotal role of training in the performance of PEEB on bird classification tasks. We demonstrate that without adequate tuning, the results are indistinguishable from random chance.

Experiment We conduct the experiment based on OWL-ViT$_{\text{base32}}$. We retain all components as illustrated in Figure 6.25, with one exception: we substitute the Part MLP with the MLP layer present in the box prediction head of OWL-ViT because the proposed layers require training. The MLP layers in the box prediction head project the part embeddings to match the dimensionality of the text embeddings. Our focus is on assessing the classification accuracy of the untuned PEEB on two datasets: CUB and NABirds.

Results Table 6.14 reveals the outcomes of our experiment. Without training, PEEB yields classification accuracies of 0.55% for CUB and 0.31% for NABirds, both of which are proximate to random chance (0.5% for CUB and 0.1% for NABirds). However, with training, the model's performance dramatically transforms: 64.33% for CUB (an increase of +63.78 points) and 69.03% for NABirds (a leap of +68.72 points) for PEEB$_{[-\text{test}]}$. These pronounced disparities underscore the vital role of training in PEEB.

Table 6.14: Impact of Training on Classification Accuracies: Untuned PEEB yields 0.55% on CUB and 0.31% on NABirds, almost mirroring random chance. With training (PEEB$_{[-\text{test}]}$), accuracy surges by +63.78 points on CUB and +68.72 points on NABirds.

	CUB	NABirds
PEEB (no training)	0.55	0.31
PEEB$_{[-\text{test}]}$ pre-trained	64.33	69.03
PEEB$^{\text{CUB}}_{[-\text{test}]}$ finetuned	86.73	-

Failure analysis

Since PEEB has two branches, box detection, and descriptor matching, we would like to find out, in the failure case, what is the main cause. i.e., is it because of the mismatch in the descriptor to the part embeddings? Or is it because the box detection is wrong? From our ablation study, it turns out that most errors come from the descriptor-part matching.

Experiment We conduct the experiment with $\text{PEEB}_{[-\text{test}]}$ on CUB test set. Specifically, we measure the box detection accuracy based on the key point annotation in CUB dataset, i.e., We consider the box prediction as **correct** if the prediction includes the human-annotated key point. We report the box prediction error rate (in %) based on parts.

Results As shown in table 6.15, the average error rate difference between success and failure cases is merely 0.38. That is, in terms of box prediction, the accuracy is almost the same, disregarding the correctness of bird identification. It indicates that the prediction error is predominantly due to the mismatch between descriptors and part embeddings. We also noted that some parts, like Nape and Throat, have a very high average error rate, which may greatly increase the matching difficulties between descriptors and part embeddings.

Table 6.15: Error rate of Box Prediction in Failure and Success Cases. We report the box prediction error rate, depending on whether the prediction box includes ground truth key points. No major difference is found between them, which means the failure is largely due to the part-descriptor mismatch.

Body Part	**Average**	**Back**	**Beak**	**Belly**	**Breast**	**Crown**	**Forehead**	**Eyes**	**Legs**	**Wings**	**Nape**	**Tail**	**Throat**
Failure Cases	16.52	23.38	3.28	8.06	15.96	7.41	24.72	7.29	5.63	3.36	64.79	7.25	27.07
Success Cases	16.14	23.03	2.96	7.44	18.64	7.13	21.53	3.93	6.85	2.68	68.66	6.40	24.38
Difference	**0.38**	0.35	0.33	0.62	-2.68	0.28	3.19	3.36	-1.22	0.68	-3.87	0.85	2.68

Evaluation of predicted boxes from PEEB

Our proposed method primarily aims to facilitate part-based classification. While the core objective is not object detection, retaining the box prediction component is paramount for ensuring model explainability. This section delves into an evaluation of the box prediction performance of our method against the OWL-ViT$_{\mathsf{base32}}$ model.

Experiment Given our focus on part-based classification, we aimed to ascertain the quality of our model's box predictions. To this end, we employed two metrics: mean Intersection over Union (IoU) and precision based on key points. We opted for mean IoU over the conventional mAP because: (1) Ground-truth boxes for bird parts are absent, and (2) our model is constrained to predict a single box per part, ensuring a recall of one. Thus, we treat OWL-ViT$_{\mathsf{large}}$'s boxes as the ground truth and evaluate the box overlap through mean IoU. Furthermore, leveraging human-annotated key points for bird parts, we measure the precision of predicted boxes by determining if they contain the corresponding key points. We evaluate our finetuned models on their corresponding test sets. For instance, PEEB$^{\mathsf{Akata}}_{[-\mathsf{cub}]}$, finetuned based on the CUB split [3], is evaluated on the CUB test set.

Results Our evaluation, as presented in table 6.16, shows that PEEB's box predictions do not match those of OWL-ViT$_{\mathsf{base32}}$. Specifically, on average, there is a -5 to -10 points reduction in mean IoU for CUB and NABirds datasets, respectively. The disparity is less distinct when examining precision based on human-annotated key points; our method records about -0.14 points lower precision for CUB and -3.17 points for NABirds compared to those for OWL-ViT$_{\mathsf{base32}}$. These observations reinforce that while PEEB's box predictions might not rival these dedicated object detection models, they consistently highlight the same parts identified by such models as shown in figure 6.30. It is important to note that our approach utilized the same visual embeddings for both classification and box prediction tasks. This alignment emphasizes the part-based nature of our model's predictions.

Table 6.16: Model evaluation on CUB and NABirds test sets. We evaluate the predicted boxes on two *ground-truth* sets; (1) predicted boxes from OWL-ViT$_{\text{large}}$ as ground-truths, and (2) OWL-ViT$_{\text{large}}$'s boxes that include the human-annotated key points. Our method has slightly lower performance in terms of mean IoU but comparable precision.

	Models	Mean IoU		
		(1) All	(2) w/ Keypoints	Precision
CUB	OWL-ViT$_{\text{large}}$	**100.00**	**100.00**	**83.83**
	OWL-ViT$_{\text{base32}}$	44.41	49.65	83.53
	PEEB (Average)	35.98	40.14	83.39
	PEEB$^{\text{CUB}}_{[-\text{test}]}$	37.45	41.79	81.55
	PEEB$^{\text{Akata}}_{[-\text{cub}]}$	35.11	39.14	82.72
	PEEB$^{\text{SCS}}_{[-\text{cub}]}$	35.77	39.96	84.89
	PEEB$^{\text{SCE}}_{[-\text{cub}]}$	35.58	39.67	84.38
NABirds	OWL-ViT$_{\text{large}}$	**100.00**	**100.00**	**85.01**
	OWL-ViT$_{\text{base32}}$	40.14	47.63	83.89
	PEEB (Average)	36.47	42.01	80.72
	PEEB$^{\text{SCS}}_{[-\text{nab}]}$	36.45	42.03	80.09
	PEEB$^{\text{SCE}}_{[-\text{nab}]}$	36.49	41.99	81.34

6.3.6 Study on GPT-4 generated descriptors

Assessment of the generated part-based descriptors

We test GPT-4V on the CUB test set using the generated descriptors of 200 classes to assess their usability. Specifically, we feed GPT-4V with each test image encoded in the payload and 200 sets of part-based descriptors through a carefully designed prompt (Table 6.17). GPT-4V is asked to output one of 200 provided class names to compute the classification accuracy. As a result, GPT-4V achieves 69.4% accuracy which is slightly higher than PEEB's generalized zero-shot accuracy (64.33%) and significantly lower than PEEB results after finetuning (86-88%).

Table 6.17: Prompt for GPT-4V evaluation on CUB where *{list_of_200_classes}* is replaced by the actual 200 CUB classes while *{descriptors}* is replaced by the actual descriptors associated with a given bird image from the CUB test set.

> You are an image classifier which can tell what type of a bird is from the given image and its associated part descriptors describing 12 parts of the bird. Your answer should be strictly formatted as {"prediction": "bird_class"}.
>
> where "bird_class" is one of the following 200 bird classes: {list_of_200_classes}
>
> Given the bird image and the following descriptors: {descriptors}
>
> What kind of bird is this? Let's think step by step.

Noise measurement in GPT-4 generated descriptors

In this section, we conduct an empirical analysis to quantify the noise in descriptors generated by GPT-4 for 20 different classes within the CUB dataset. To achieve this, we manually inspect each descriptor and tally the instances where at least one factual error is present. Our findings reveal that every one of the 20 classes contains descriptors with errors, and on average, 45% of the descriptors necessitate corrections. This substantial noise level underscores the need for further refinement in our work, particularly in text descriptors.

We observe a notably high error rate in descriptors on the *back* and *wings*, with approximately 60% of these containing inaccurate information (refer to Table 6.18). This could be attributed to the challenges in distinguishing between the *back* and *wings*, given that the *back* is typically positioned behind the *wings*, yet exhibits considerable variability in size and shape. Addressing all descriptor issues by revising all 11,000 fine-grained descriptors would demand a significant investment of time and resources, which is beyond the scope of the current work. As such, we identify this as an area for future research and development, aiming to enhance the quality of the Bird-11K dataset.

Table 6.18: Summary of manual inspection results for 20 classes, highlighting the need for revision in GPT-4 generated descriptors. An average error rate of 45% indicates substantial room for improvement.

	Back	Beak	Belly	Breast	Crown	Forehead	Eyes	Legs	Wings	Nape	Tail	Throat	Average
Error Rate	60	30	50	40	50	55	50	20	60	50	35	40	45

Revising descriptors improves classification accuracy

As mentioned in the limitation section, the descriptors are generated from GPT-4 and therefore noisy and incorrect. Given that PEEB accepts open vocabulary inputs for classification, a natural way to improve classification accuracy is to improve the correctness of the descriptors.

Experiment We first collect descriptors of 183 CUB classes from AllAboutBirds. We then prompt GPT-4 to revise our original descriptors by providing the collected descriptor. We revise the descriptors with the following prompt:

```
Given the following descriptors of {class name}: {AllAboutBirds descriptors}.
Can you revise the incorrect items below (if any) of this bird, return them
as a Python dictionary, and use the key as the part name for each item? If
a partś descriptor is not specifically described or cannot be inferred from
the definition, use your own knowledge. Otherwise, leave as is. Note: please
use a double quotation mark for each item such that it works with JSON format.
{Original descriptors}
```

Where `{class name}` the placeholder for the class name, `{AllAboutBirds descriptors}` is the description collected from AllAboutBirds, `{Original descriptors}` is the descriptors we used for training.

Due to the errors in the descriptors we used to train PEEB, simply replacing the descriptors with their revised version does not lead to better performance. Because the incorrect descriptors in training change the meaning of some of the phrases. For example, the belly of Bluebunting is pure blue, but the descriptors from GPT-4 is *soft, creamy white*. In addition, the GPT-4 uses the exact same descriptor in the belly for other classes, e.g., Bluebreastedquail, which should be cinnamon. BlueFrontedFlycatcher, which should be yellow. Training the same descriptors with different colors confuses the model, and the model will convey the phrase "creamy white" with a different meaning to humans. Therefore, simply changing the descriptors to their' revised version will not work. We empirically inspect the descriptors that PEEB can correctly respond to and replace the class descriptors with the revised version. Specifically, we replace the descriptors of 17 classes in CUB and test the classification accuracy on PEEB$_{[-\text{test}]}$.

Results As shown in Table 6.19, the overall accuracy increase +0.8 points. The average improvement of the revised class is around +10.8, hitting that if we have correct descriptors of all classes, we may significantly improve the classification accuracy of the pre-trained model. However, correcting all 11k class descriptors is too expensive and out of the scope of this work. We leave it as a further direction of improving the part-based bird classification.

Table 6.19: The revised descriptors result in +0.8 for PEEB$_{[-\text{test}]}$ in CUB. In particular, the average improvement among the 17 revised classes is +10.8, hinting at the large potential of our proposed model.

Descriptors	Original	Partially Revised	Avg. Improvement
PEEB$_{[-\text{test}]}$	64.33	**65.14**	10.80

6.3.7 Qualitative Inspections

Visual comparison of predicted boxes

We provide a visual comparison of the box prediction from OWL-ViT$_{\text{large}}$, OWL-ViT$_{\text{base32}}$, and PEEB in Figure 6.30. We find that despite the fact that our predicted boxes have lower mean IoU compared to OWL-ViT$_{\text{large}}$, they are visually similar to the boxes as OWL-ViT$_{\text{base32}}$.

Qualitative examples of using randomized descriptors

We visually compare M&V and PEEB based on their utilization of descriptors. (Figures 6.31 to 6.33). Specifically, we randomly swap the descriptors of the classes and then use these randomized descriptors as textual inputs to the tested models to see how they perform. We observe that the scores from M&V tend to cluster closely together. Surprisingly, M&V's prediction remains unchanged despite the inaccurate descriptors. In contrast, PEEB, when presented with randomized descriptors, attempts to identify the best match grounded on the given descriptors.

Examples of PEEB explanations for birds

Figures 6.34 to 6.36 are examples of how PEEB makes classification based on the descriptors and how it can reject the predictions made by M&V. Since we aggregate all descriptors for the final decision, even if some of them are similar in two classes, our method can still differentiate them from other descriptors. For instance, in Figure 6.34, while other descriptors are similar, PEEB can still reject chesnut – sidedwarbler thanks to the distinct features of *forehead*, *throat* and *belly*.

Examples of PEEB explanations for dogs

Figures 6.37 to 6.39 are examples of how PEEB makes classification based on the descriptors in Stanford Dogs dataset. We demonstrate that our model works well on dogs, which indicates that our proposed method is transferable to other domains while maintaining high-quality explainability as in birds.

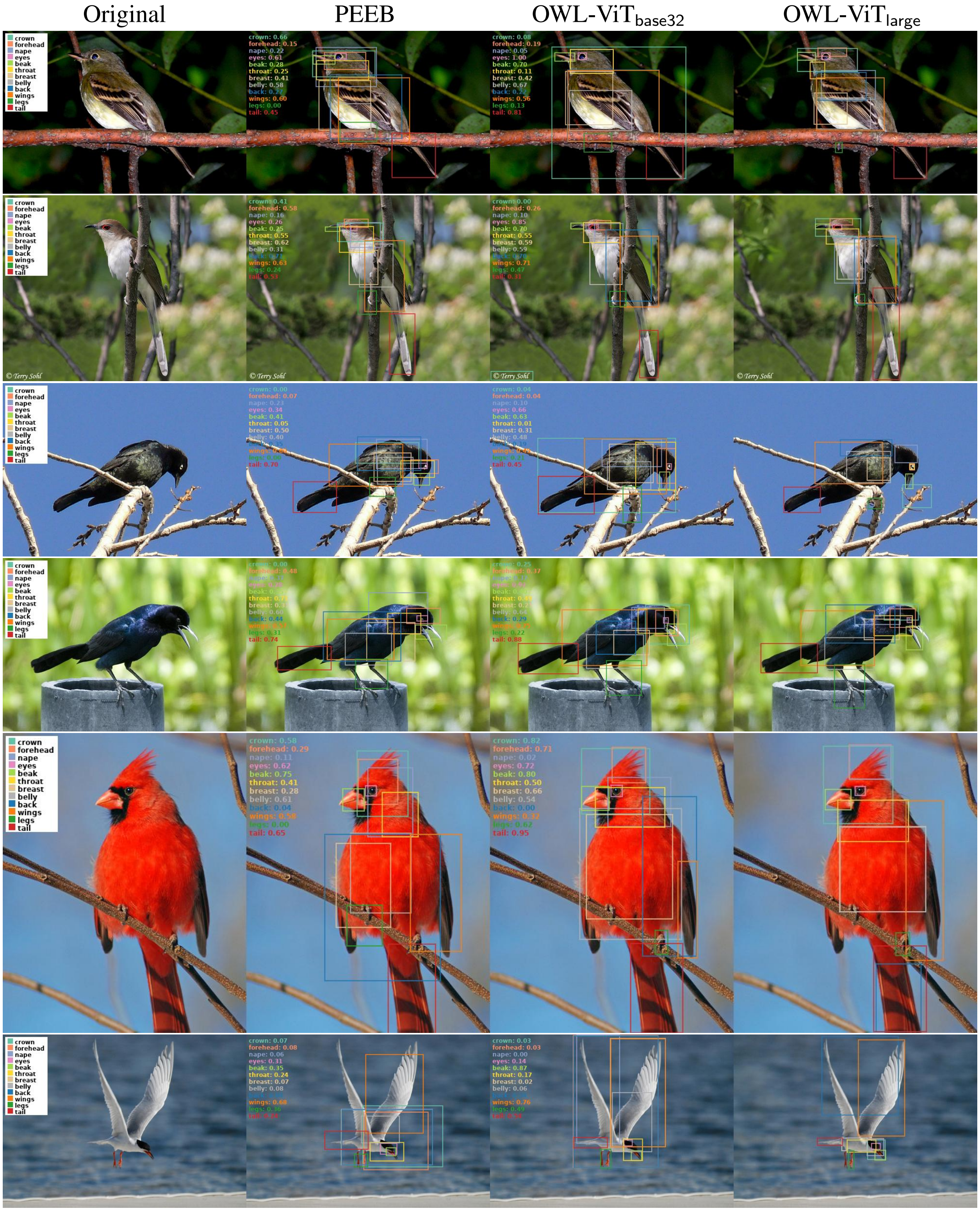


Figure 6.30: Our predicted boxes (second column) often align closely with those of OWL-ViT$_{base32}$ (third column). However, slight shifts can lead to significant IoU discrepancies. For instance, in the first row, both PEEB and OWL-ViT$_{base32}$ accurately identify the tail. Yet, variations in focus yield a stark IoU contrast of 0.45 versus 0.81.

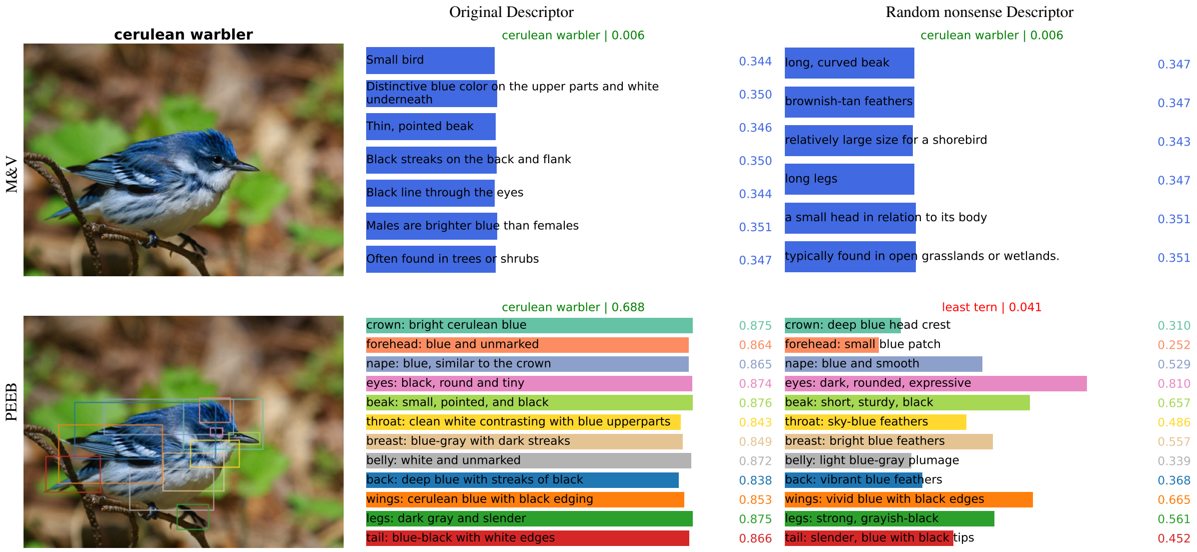


Figure 6.31: Qualitative example of original descriptors vs. randomized descriptors. Upon swapping descriptors randomly, the prediction outcomes from M&V exhibit minimal variations.

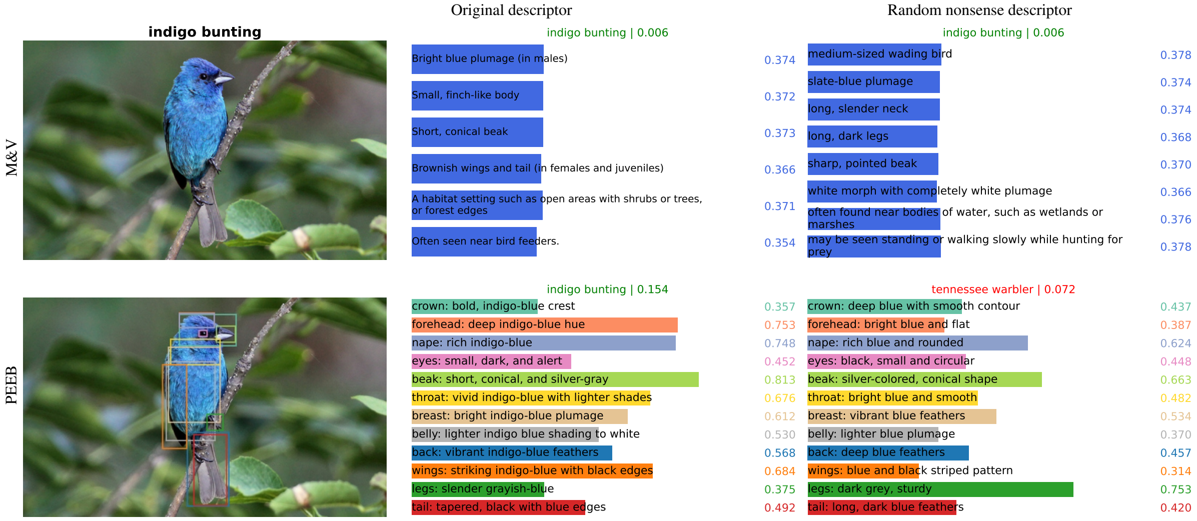


Figure 6.32: Qualitative example of original descriptors vs. randomized descriptors. Since PEEB's decision is made by the descriptors, the model will try to find the descriptors that best match the image. e.g., in the random descriptors, most parts are blue.

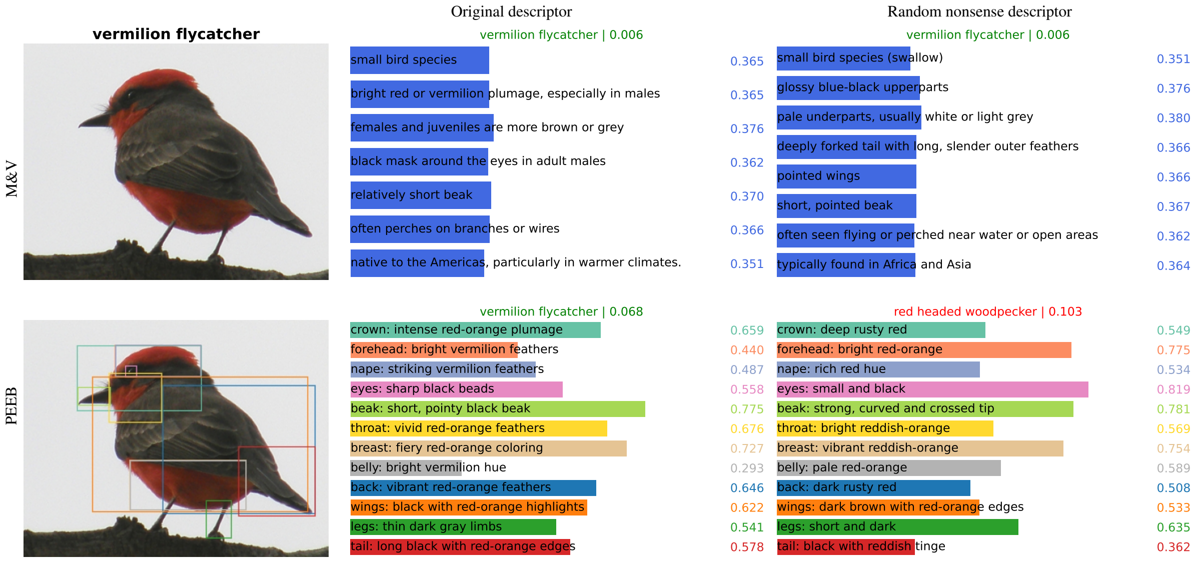


Figure 6.33: Qualitative example of original descriptors vs. randomized descriptors. M&V maintains similar scores even for mismatched descriptors. For instance, "bright red or vermilion plumage, especially in males" receives a score lower than "glossy blue-black upperparts". Conversely, PEEB leverages the descriptors for classification, consistently relying on the descriptors that most closely align with the image.

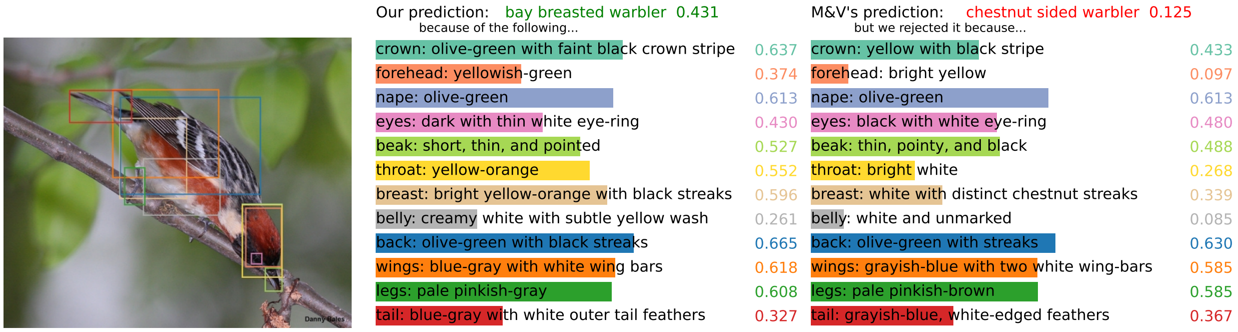
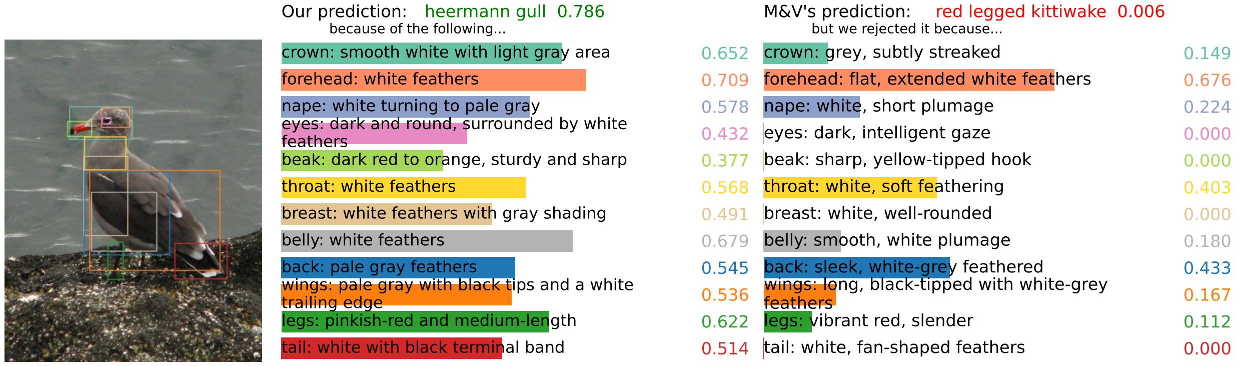
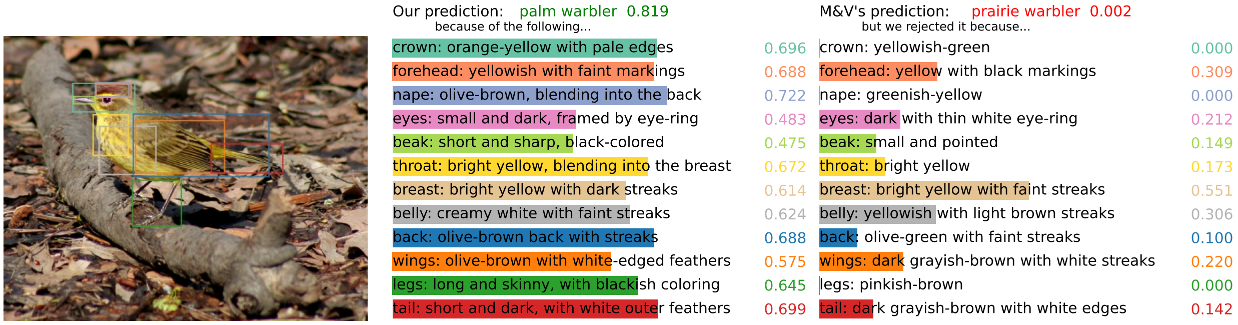


Figure 6.34: An example of PEEB explanation. We can see that the descriptors of these two classes are largely similar, but PEEB makes the correct prediction based on the distinctive feature of the forehead in the two classes.



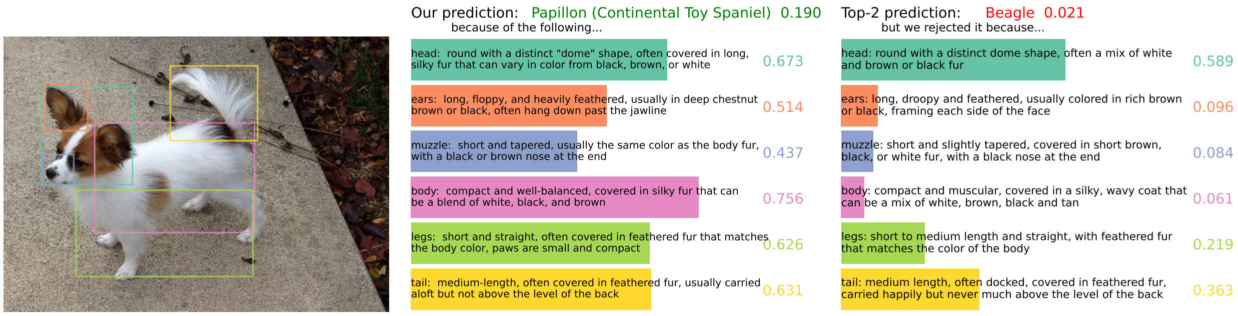
Our prediction: heermann gull 0.786 because of the following...		M&V's prediction: red legged kittiwake 0.006 but we rejected it because...	
crown: smooth white with light gray area	0.652	crown: grey, subtly streaked	0.149
forehead: white feathers	0.709	forehead: flat, extended white feathers	0.676
nape: white turning to pale gray	0.578	nape: white, short plumage	0.224
eyes: dark and round, surrounded by white feathers	0.432	eyes: dark, intelligent gaze	0.000
beak: dark red to orange, sturdy and sharp	0.377	beak: sharp, yellow-tipped hook	0.000
throat: white feathers	0.568	throat: white, soft feathering	0.403
breast: white feathers with gray shading	0.491	breast: white, well-rounded	0.000
belly: white feathers	0.679	belly: smooth, white plumage	0.180
back: pale gray feathers	0.545	back: sleek, white-grey feathered	0.433
wings: pale gray with black tips and a white trailing edge	0.536	wings: long, black-tipped with white-grey feathers	0.167
legs: pinkish-red and medium-length	0.622	legs: vibrant red, slender	0.112
tail: white with black terminal band	0.514	tail: white, fan-shaped feathers	0.000

Figure 6.35: An example of PEEB explanation. M&V incorrectly classifies it as red – leggedkittiwake where the heermanngull does not have red legs but a red beak. This example shows that CLIP is strongly biased towards some particular descriptors.



Our prediction: palm warbler 0.819 because of the following...		M&V's prediction: prairie warbler 0.002 but we rejected it because...	
crown: orange-yellow with pale edges	0.696	crown: yellowish-green	0.000
forehead: yellowish with faint markings	0.688	forehead: yellow with black markings	0.309
nape: olive-brown, blending into the back	0.722	nape: greenish-yellow	0.000
eyes: small and dark, framed by eye-ring	0.483	eyes: dark with thin white eye-ring	0.212
beak: short and sharp, black-colored	0.475	beak: small and pointed	0.149
throat: bright yellow, blending into the breast	0.672	throat: bright yellow	0.173
breast: bright yellow with dark streaks	0.614	breast: bright yellow with faint streaks	0.551
belly: creamy white with faint streaks	0.624	belly: yellowish with light brown streaks	0.306
back: olive-brown back with streaks	0.688	back: olive-green with faint streaks	0.100
wings: olive-brown with white-edged feathers	0.575	wings: dark grayish-brown with white streaks	0.220
legs: long and skinny, with blackish coloring	0.645	legs: pinkish-brown	0.000
tail: short and dark, with white outer feathers	0.699	tail: dark grayish-brown with white edges	0.142

Figure 6.36: An example of PEEB explanation. We can see that when the descriptor does not match the image, the matching score tends to be zero, e.g., *crown: yellowish-green*. The clear differences in scores provide us transparency of the model's decision.



Our prediction: Papillon (Continental Toy Spaniel) 0.190 because of the following...		Top-2 prediction: Beagle 0.021 but we rejected it because...	
head: round with a distinct "dome" shape, often covered in long, silky fur that can vary in color from black, brown, or white	0.673	head: round with a distinct dome shape, often a mix of white and brown or black fur	0.589
ears: long, floppy, and heavily feathered, usually in deep chestnut brown or black, often hang down past the jawline	0.514	ears: long, droopy and feathered, usually colored in rich brown or black, framing each side of the face	0.096
muzzle: short and tapered, usually the same color as the body fur, with a black or brown nose at the end	0.437	muzzle: short and slightly tapered, covered in short brown, black, or white fur, with a black nose at the end	0.084
body: compact and well-balanced, covered in silky fur that can be a blend of white, black, and brown	0.756	body: compact and muscular, covered in a silky, wavy coat that can be a mix of white, brown, black and tan	0.061
legs: short and straight, often covered in feathered fur that matches the body color, paws are small and compact	0.626	legs: short to medium length and straight, with feathered fur that matches the color of the body	0.219
tail: medium-length, often covered in feathered fur, usually carried aloft but not above the level of the back	0.631	tail: medium length, often docked, covered in feathered fur, carried happily but never much above the level of the back	0.363

Figure 6.37: An example of PEEB explanation for dogs. Like birds, PEEB first identifies the predefined parts and then matches them to the descriptions.

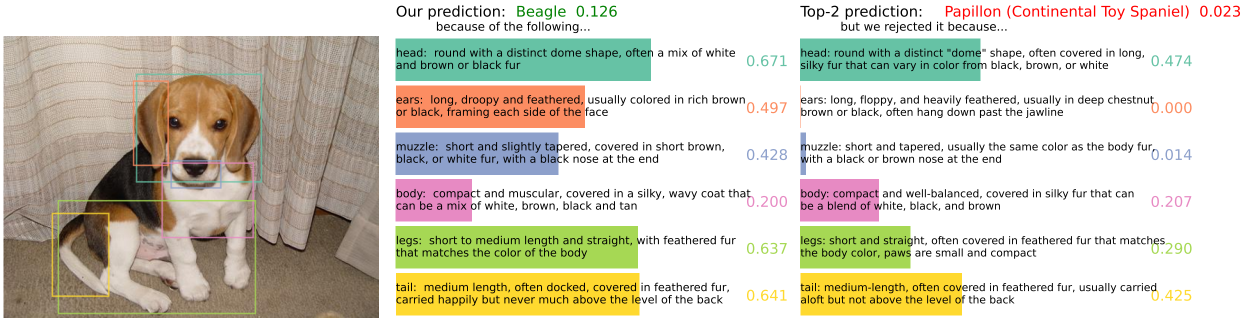


Figure 6.38: An example of PEEB explanation for dogs. Like birds, PEEB first identifies the predefined parts and then matches them to the descriptions.

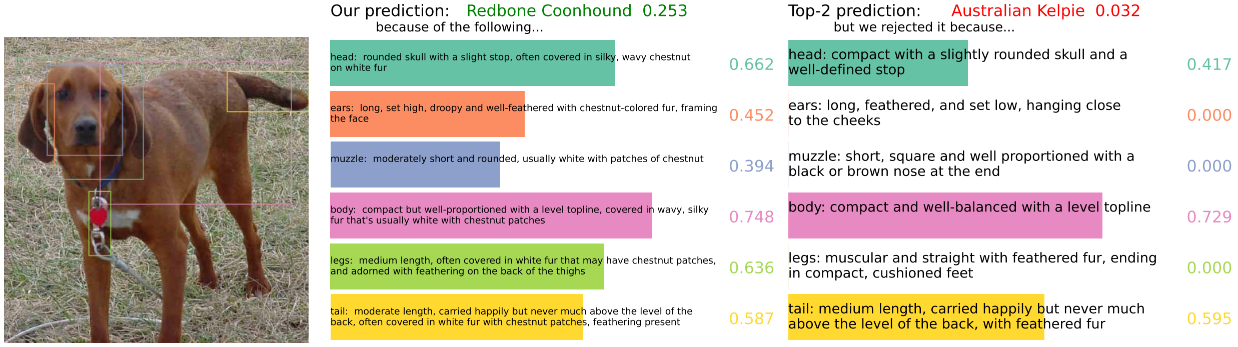


Figure 6.39: An example of PEEB explanation for dogs. Like birds, PEEB first identifies the predefined parts and then matches them to the descriptions.