Contributions of Shape, Texture, and Color in Visual Recognition (Appendix)

Yunhao Ge*, Yao Xiao*, Zhi Xu, Xingrui Wang, and Laurent Itti

University of Southern California
https://github.com/gyhandy/Humanoid-Vision-Engine

Appendix

A Details of Human Vision Engine (HVE)

A.1 Image Parsing and Foreground Identification

As described in the main paper Sec. 3.1, we use entity segmentation and foreground object identification to simulate the preprocessing behavior of the human vision system. An illustration is shown in Fig. 1.

The entity segmentation can parse an input image and output a set of binary images masks. Each mask represents a single object or stuff in the image. For an input raw image $I_{\text{raw}} \in \mathbb{R}^{H\times W\times C}$, we denote the image mask sets as $\{I_{\text{mask},k}\}$, where $k = 1, 2, ...,$ denotes each different object. For each image mask $I_{\text{mask},k}$, pixel $I^{(i,j)}_{\text{mask},k}$ equals to 1 if and only if pixel $I^{(i,j)}_{\text{raw}}$ belongs to objects k, otherwise equals to 0.

For foreground identification, we borrow the learned knowledge from a pretrained model which already learn the foreground class. As Grad-CAM [4] could generate class-specific saliency map M_{cam} given the model's prediction, which represent how important each pixel contribute to the specific prediction. So the pixel with a higher activation value is more likely belong to foreground. We first use a pretrained model to generate a saliency map M_{cam} with Grad-Cam, and then we can get the binary attention map M_{att} based on M_{cam}. For mask M_{att}, pixel $M^{(i,j)}_{\text{att}}$ equals to 1 if and only if pixel $M^{(i,j)}_{\text{cam}} > \tau$ (we set τ to be the median value of M_{cam}), otherwise equals to 0.

After getting M_{att}, we can compute Intersection over union (IoU) score of each image mask $I_{\text{mask},k}$,

$$S_k = SUM(I_{\text{mask},k} \cap M_{\text{att}})/SUM(I_{\text{mask},k})$$

Where $SUM()$ calculates the number of pixels. Then we choose the one with the highest score as the foreground mask.

* Yunhao Ge and Yao Xiao contributed equally

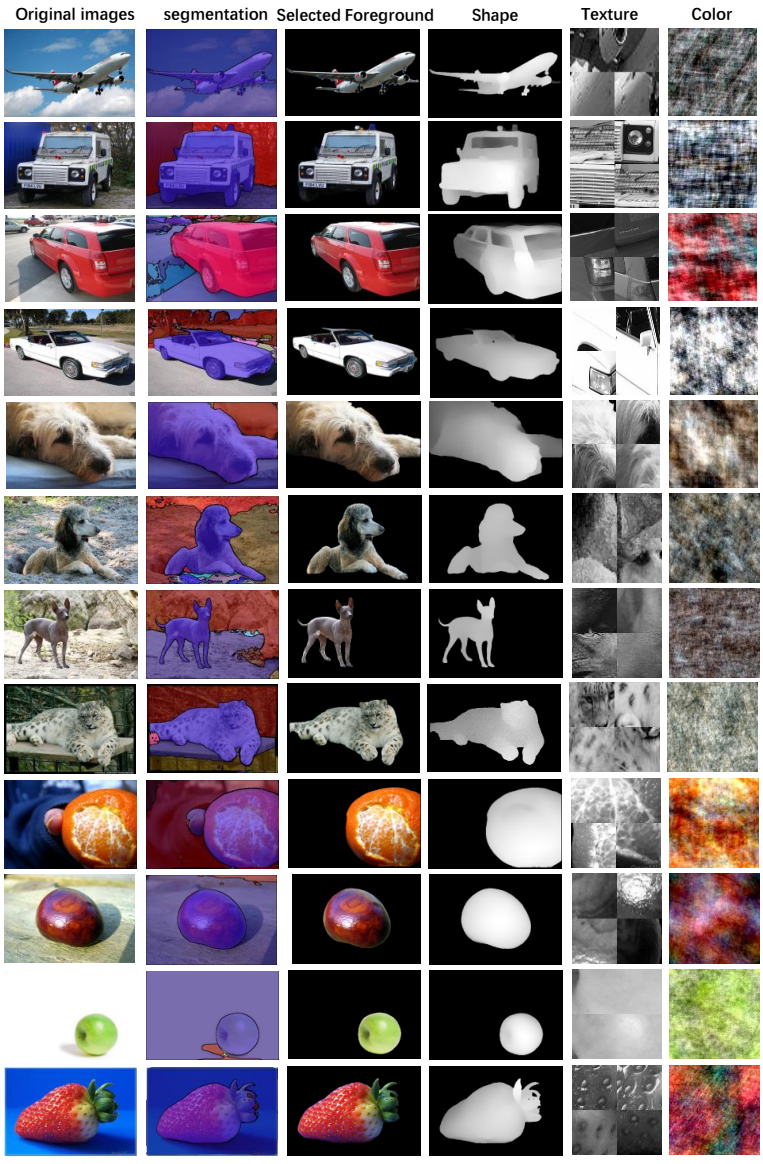

Fig. 1: Example of the preprocessing result and feature extraction.

Fig. 2: The process for extracting texture. We cut the image of the foreground object into several square patches as shown in (a) and select the patch pool as shown in (b). (c) is the final texture feature, the concatenation of k randomly selected patches from the patches pool.

A.2 Feature Extractor

In HVE, with the preprocessed image $I \in \mathbb{R}^{H \times W \times C}$, we use the three independent feature extractors to obtain the corresponding image feature: shape image (I_s), texture image (I_t), color image (I_c). Here, we will introduce more details about the texture and color extractor.

Texture Extractor Fig. 2 visualizes the process of extracting texture. First, to remove the color information, we convert the RGB object segmentation to a grayscale image. Since we want to get rid of the influence of background and extract the local texture feature as well as the global texture feature, we compute the maximum circumscribed rectangle of the object by its 2D mask and resize this rectangle part to get a 224×224 image. In this way, we can eliminate the most background and focus on the object. Next, we will cut this new image into several square patches. Specifically, We first cut this image into several square patches, as shown in Fig. 2 (a), if the overlap ratio between the patch and the original 2D object segment is larger than a threshold τ, we will add them to a patch pool (we set τ to be 0.99 in our experiments, which means the over 99% area of the patch belong to the object), as shown in Fig. 2 (b). Since we want to extract both local texture information (one patch) and global texture information (whole image), we randomly select k patches from the patch pool and concatenate them to a new texture image (I_t). We set $k = 4$ because we

want to concatenate those patches to a square image, which means k should be a square number. If we set $k = 1$, we can get the maximum square patch of the object but we lose some small, local texture. If we set $k = 9$, the patch will be too small to contain useful information. To dynamically fit the object size and minimize the texture information loss. The number of patches we cut from the original image is dynamic. We will first cut the image into 9 square patches, if we can get $k = 4$ valid patches that can be added into the patch pool from these 9 patches, we will stop processing this image. However, if we cannot get 4 valid patches, we will cut the original images into 16, 25, 36... smaller patches until we can get 4 valid patches. After concatenation, we can get the texture image I_t, such as Fig. 2 (c). More results are shown in Fig. 1.

Color Extractor We have two different ways to extract the color feature I_c, phase scrambling, and color blocks.

Phase scrambling For a given image $I \in \mathbb{R}^{H\times W\times C}$ and a random matrix $z \in \mathbb{R}^{H\times W}$, we use the 2D fast fourier transform (FFT) on channel j of the image and get the output $I_j^* \in \mathbb{C}^{H\times W}$, where $j = 1, 2, ..., C$. We then caculate their modulus r_j and angle θ_j. Similarly, we apply the FTT to the random matrix z and get the transformed result $z^* \in \mathbb{C}^{H\times W}$ and its angle φ. With r_j, θ_j, φ, we can construct a new complex variable $T_j = r \cdot e^{i(\theta+p\varphi)}$, where $p \in [0, 1]$ is a scramble factor. After the mapping backing by 2D inverse fast fourier transform and rescale the result to range $[0, 255]$, the output will be the channel j of our color feature $I_c \in \mathbb{R}^{H\times W\times C}$. This process can be described formally as:

$$I_j^* = \mathrm{FFT}(I_j) = r_i \cdot e^{i\theta_j} \tag{1}$$

$$z^* = \mathrm{FFT}(z) = s \cdot e^{i\varphi}, \tag{2}$$

$$T_j = r \cdot e^{i(\theta_j+p\varphi)}, \tag{3}$$

$$I_{c,j} = \mathrm{rescale}(\mathrm{IFFT}(T_j)), \tag{4}$$

where $j = 1, 2, ..., C$ and $I_{c,j}$ is the channel j of I_c. More results are shown in Fig. 1.

Color Blocks The second method uses statistical color histogram representation [7,2]. The original RGB color space is a three dimensional cubic $\{(r, g, b)|r, g, b \in [0, 255]\}$. In order to represent the distribution of color for each image, we first choose 27 center points which are uniformly distributed in the entire color space. The colors we choose are shown in Fig. 3. For an input image, we assign each pixel to its closest center point by calculating their manhattan distance. By counting how many pixels belong to each color center and calculating the percentage, we can summarize the result to a color block image of size 224×224 as our color feature I_c (the examples are shown in Fig. 4). The color block consists of various stripes in different widths and colors. The color of stripes comes from the color center and the width represents how many percent of pixels this center covered in the input image. For instance, if there are 10% pixels assign to white

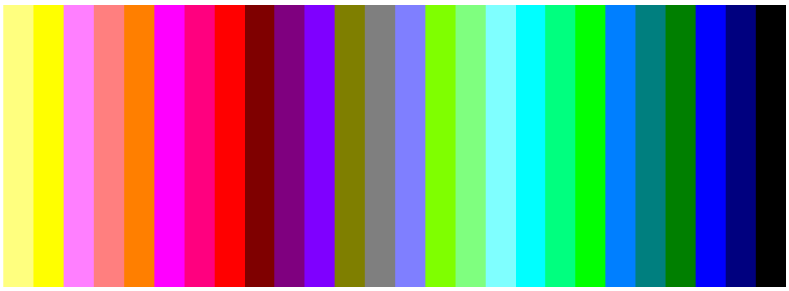

Fig. 3: 27 Color centers for the color block.

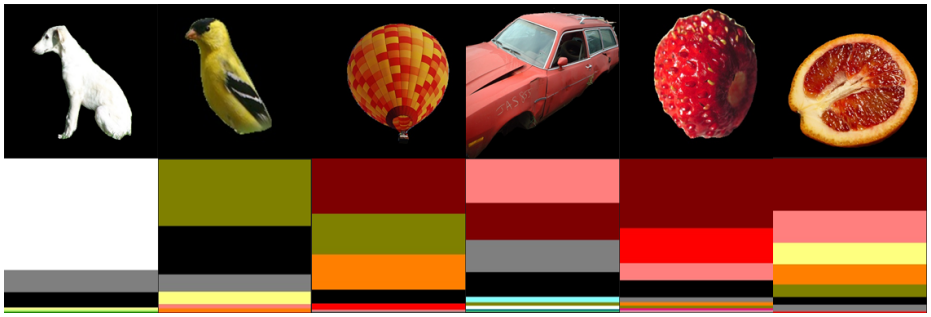

Fig. 4: Example of color block representation. The first row is original images, while the second row is the color blocks of those images.

and 90% pixels of black, we will generate a image whose 10% pixels are RGB $(255, 255, 255)$ and 90% pixels are RGB $(0, 0, 0)$. This image (I_c) will not contain any shape or texture information. Fig. 4 shows some examples of the color blocks.

Compared with phase scrambling, this method is more intuitive to understand but may lose information when approximating RGB value in color space to color centers.

A.3 Details about the Humanoid Neural Network

Table 1: Architecture of the humanoid neural network. N is the number of labels, depending on which dataset is being used.

Layer	Input → Output Shape	Layer Information
Shape Encoder	(224, 224, 1) → (7, 7, 512)	ResNet18
Texture Encoder	(224, 224, 1) → (7, 7, 512)	ResNet18
Color Encoder	(224, 224, 3) → (7, 7, 512)	ResNet18
Concatenation Layer	3×(7, 7, 512) → (7, 7, 1536)	-
Pooling Layer	(7, 7, 1536) → (1, 1, 1536)	Average Polling-(k7x7, s1, p0)
Flatten Layer	(1, 1, 1536) → 1536	-
Hidden Layer	1536 → 512	Linear, ReLU
Output Layer	512 → N	Linear, Softmax

The three encoders (E_s, E_t, E_c) use ResNet18 as backbone. E_s takes shape feature with size $224 \times 224 \times 1$ as input, while E_t takes texture feature with size $224 \times 224 \times 1$ and E_c takes color feature with size $224 \times 224 \times 3$. They all produce the feature with size $7 \times 7 \times 512$. When training the encoders, these output features are then passed into a fully connected layer, which output the vectors with length N, the number of classes in the dataset.

During training the interpretable aggregation module, we freeze the three encoders and concatenate their output features along the channel dimension into a tensor of size $7 \times 7 \times 1536$. This tensor is the input of the interpretable aggregation module, which is a two-layer MLP. The final output is a vector with length N, the number of classes in the dataset. The summary of the structure of the three encoders and the aggregation module is shown in Table 1.

We use Adam optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and set batch size to 64, learning rate is 0.001.

B Experiments Details

B.1 Influence of Random Selection in Texture Feature Extractor

We re-ran 10 times main paper Table.2 using different random seeds for texture (patch selections and shuffling the order sequence). The mean and std shown in Table. 2.

Table 2: Influence of Random Selection in Texture Feature Extractor.

contribution	Shape	Texture	Color
Shape-bias DS	47.1% ± 2.7%	34.5% ± 2.2%	18.4% ± 1.4%
Texture-bias DS	5.2% ± 1.8%	62.4% ± 2.3%	32.3% ± 1.6%
Color-bias DS	12.4% ± 2.1%	19.1% ± 1.9%	68.5% ± 3.6%

B.2 More quantitative contribution summary results of Humanoid NN

To further evaluate the contribution of shape texture and color, for each feature V (e.g., shape feature Vs), we compute the accuracy if we only use the rest features (e.g., combine texture Vt and color Vc) as input, and calculate the accuracy drop compared with using all three features as input (last column in Table 1). That accuracy drop represents the "unsubstitutability" or the "necessity" or "non-redundant contribution" of that feature for visual recognition. The results (Table. 3) are consistent with our previous results (main paper Table 1) on the contribution/importance of each feature. For example, in the shape biased dataset, the largest accuracy drop is when we remove the shape feature.

Table 3: More quantitative contribution summary results of Humanoid NN

accuracy	Shape + Texture	Shape + Color	Texture + Color	**all**
Shape biased DS	94%	92%	86%	95%
Texture biased DS	89%	80%	83%	90%
Color biased DS	83%	88%	87%	92%

B.3 Different interpretable aggregation module

We conducted experiments to substitute the non-linear MLP with simple average pooling followed by an output classification layer. Contributions are shown in table. 4. While the numerical results differ, the ordering and conclusions remain (e.g., shape texture is most important in the shape-biased dataset). The experiments results show that the contribution result is robust to the aggregation module.

Table 4: Average pooling interpretable aggregation module

Ave-Pool	Shape ratio	Texture ratio	Color ratio
Shape biased DS	48%	40%	12%
Texture biased DS	1%	79%	20%
Color biased DS	1%	4%	95%

B.4 Sample Question of Human Experiments

We designed human experiments that asked participants to classify reduced images with only shape, texture, or color features. We publicly posted our questionnaires on the internet and sent the questionnaire link to the students and machine learning researchers in different universities. The reduced image contained only shape, texture, or color information. To make sure that the participants understood the class definitions well, we also showed them two example images of each class at the bottom of the screen with the assigned class label written below (for instance, some participants may not have been familiar with what the "beach wagon" ImageNet class is). Here, as shown in Fig. 5, we demonstrate a screenshot of an experiment trial in our experiments.

B.5 Contributions of Features in Different Tasks

Table 5: Local bias for CUB classes.

ratio	shape	texture	color
Bewick Wren	**44.8%**	21.3	33.8%
Gadwall	**43.6%**	24.2%	32.2%
Brown Pelican	**41.6%**	33.0%	25.4%
American Pipit	6.8%	**72.7%**	20.4%
Eared Grebe	5.6%	**70.3%**	24.1%
Harris Sparrow	13.5%	**64.1%**	22.4%
Kentucky Warbler	0.1%	0.7%	**99.2%**
Cape May Warbler	0.3%	1.2%	**98.5%**
Gray crowned Rosy Finch	1.1%	1.5%	**97.4%**

Fig. 6 shows more example images of the iLab dataset [1]. In the figure, we can see that the military vehicles are always in the unique green, which matches with the result in our experiment that color is the most discriminative feature to classify military among other vehicles.

We also compute local bias for each class in CUB dataset [8]. For each feature among shape, texture, and color, we select the top 3 classes whose feature

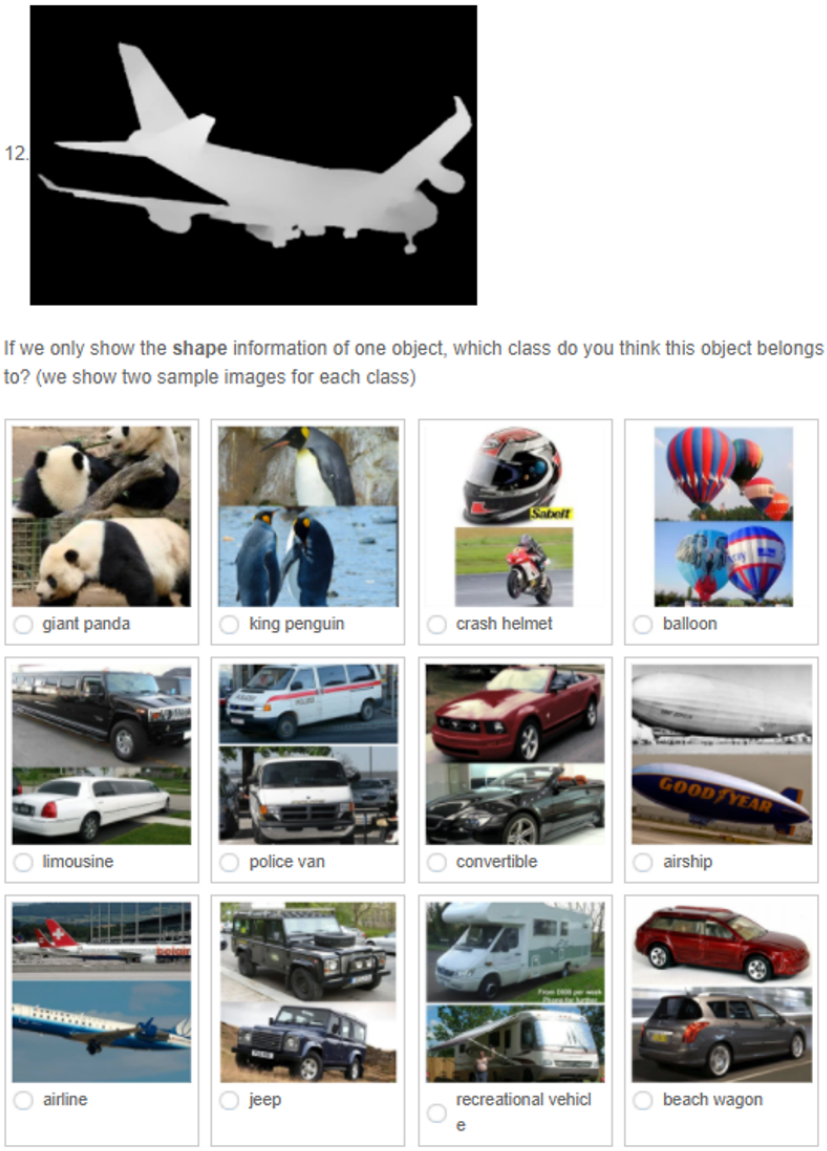

Fig. 5: A screenshot of an experiment trial in our human experiments.

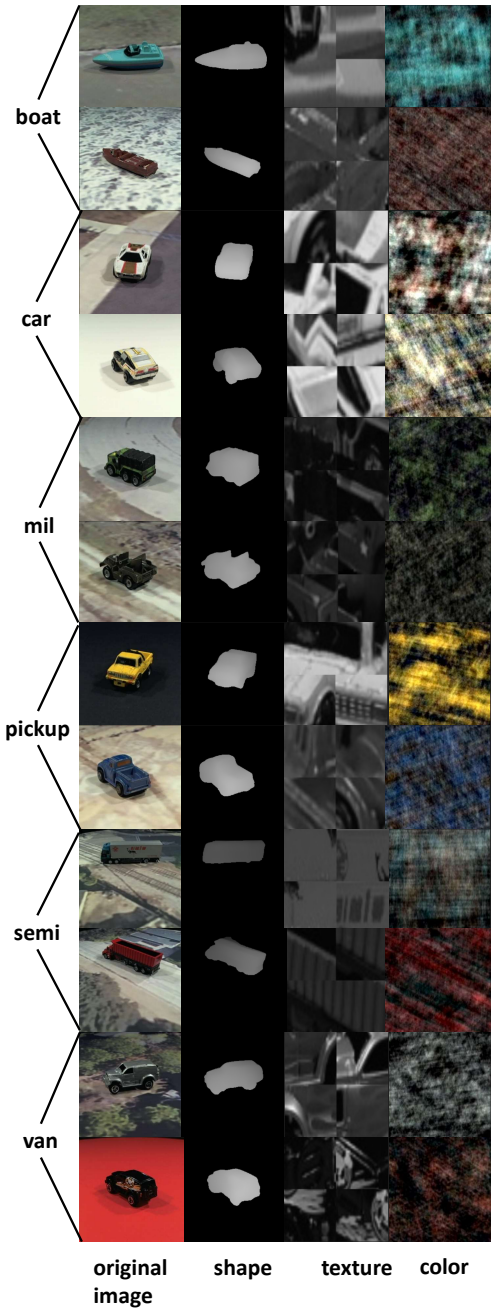

Fig. 6: Example images for ilab-20M classes.

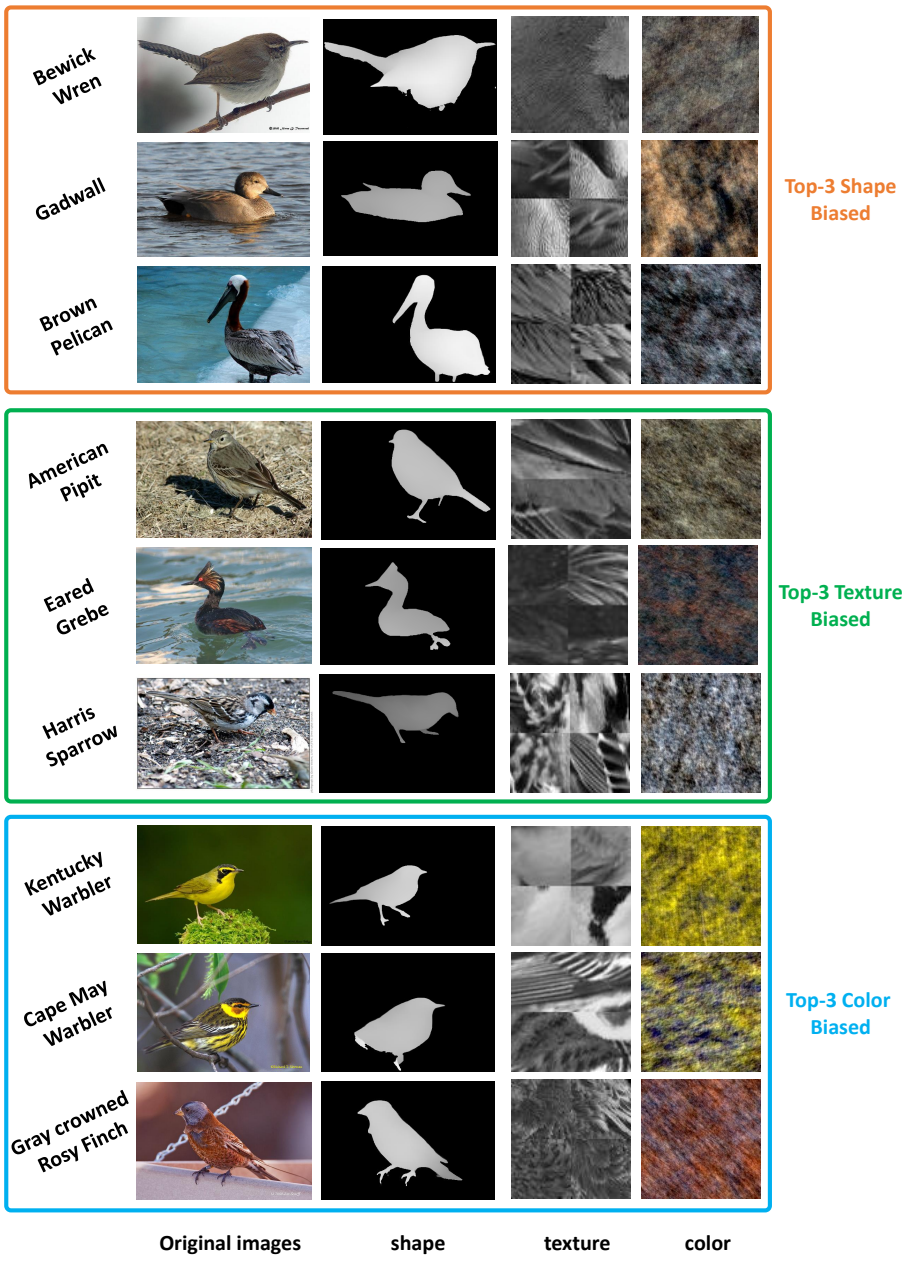

Fig. 7: Example images for CUB classes. The first 3 rows show the top-3 classes in CUB whose shape contribution is highest; The middle 3 rows show top-3 classes whose texture contribution is highest; The last 3 rows show top-3 classes whose color contribution is highest. Table. 5 shows the quantity biased results for these classes.

contributions are the highest, and we show the bias for these 9 classes in Table. 5. To show more details of these top classes for each feature, we show the example images in Fig. 7. For the first three-row classes, the shape is the most discriminative feature to classify them among other birds; for the middle three-row classes, the texture is the most discriminative feature, while color is the most discriminative feature to classify the last three-row classes.

C Details of Humanoid Application

C.1 Process of Open-world Zero-shot Learning

Class	zebra	fowl	wolf	sheep	apple
shape root	horse, donkey	turkey, goose cock	hyena, fox, tiger	goat, bull	tomato, orange, pear
texture root	tiger, piano keys	turkey, penguin	fox, dog	dog, goat	cherry, tomato
color root	panda, penguin	turkey	dog	elephant, goat	cherry, tomato

Table 6: More open-world zero-shot learning result.

As describe in main paper Sec.5.1, we conduct open-world zero-shot learning with HVE. Here, we describe more details about the experiment.

In step 1, to represent learnt knowledge, we use feature extractors E_s, E_t, E_c to get the shape, texture, and color representation $V_s^{(k,i)}$, $V_t^{(k,i)}$, $V_c^{(k,i)}$ of the i_{th} image of seen class k. Then, given an unseen class image I_{un}, we use the same feature extractors to get its feature-wise representation V_s', V_t', V_c'. To retrieve learnt classes as description, we calculate the average distance d_m^k between I_{un} and images of other class k in the latent space on feature m, described formally as

$$d_m^k = \frac{1}{n_k} \sum_{i \in \mathcal{T}_k} d_m^{(k,i)} = \frac{1}{n_k} \sum_{i \in \mathcal{T}_k} ||V_m' - V_m^{(k,i)}||_2, \tag{5}$$

where $m \in \{s, t, c\}$ is "shape", "texture" or "color", $\mathcal{T}_k$ represents images of seen class k in the training set and n_k is the number of images in class k. For example, given a wolf image, we can get $R_s = \{\text{hyena}, \text{fox, tiger}\}$, $R_t = \{\text{fox}, \text{dog}\}$, $R_c = \{\text{dog}\}$. We provide more examples in table.6.

After we get $R_s = \{\text{hyena}, \text{fox, tiger}\}$, $R_t = \{\text{fox}, \text{dog}\}$, $R_c = \{\text{dog}\}$, we use ConceptNet [6] and word embedding to predict its label (Step 2 in main paper Sec. 5.1). Specifically, if we directly use the candidate pool got from ConceptNet and calculate their ranking score, the top 5 results are animal, four-legged animal,

mammal, wolf, fox. We can find wolf are the first concrete class we can obtain, which achieved our open-world zero-shot classification.

To show the quantitative results of our method, we realize that, although the first 3 results (animal, four-legged animal, mammal) in the previous wolf example is somewhat correct, we want to get a more concrete answer which is a wolf. Thus, we design a fixed candidate which excludes these broad words and only contains unseen classes and some disturbances. In this way, we can get a quantitative accuracy without the influence of those broad words like animal, fruit, bird.

Here, we provide the experiment setting of our seen class list and our candidate pool. Our seen classes list totally contain 36 classes, they are horse, tiger, panda, penguin, piano keys, cheetah, hyena, dog, lemon, koala, fur, squirrel, fox, rabbit, goose, sea lion, elephant, otter, duck, cock, chimpanzee, goat, orange, ball, bull, tomato, cherry, pear, turkey, seal, porpoise, alpaca, pigeon, lion, donkey, bear. Our candidate pool totally contains 20 classes which are fowl, zebra, bear, wolf, husky, swan, giraffe, jackal, peach, sheep, seal, apple, banana, train, bag, balloon, car, pen, table, eagle. The results are shown in the main paper Table 5.

C.2 Detail of Prototypical network

As described in the main paper Sec.5.1, we conduct one-shot learning using the prototypical network [5]. Here, we provide more details about the experiment. Prototypical network use the same training and test set with HVE. It is not clear for a prototypical networks to direct conduct zero-shot in the released code, so we provide a easier mode, one-shot for each test class, and use its one-shot setting. To train the prototypical network, we set the input image size as $3 \times 224 \times 224$, hidden dimension as 64, learning rate as $1e-3$. We use their official code and follow the same training strategy as [5]. During testing, we use 5-way 1-shot setting, table.5 in the main paper provide the final result of the prototypical network.

C.3 Cross-features retrieval accuracy

Here we report the cross feature retrieval accuracy described in Sec. 5.2 in the main paper.

Table 7: Cross-features retrieval accuracy on biased datasets (DS).

input	shape			texture			color		
retrieval	shape	texture	color	shape	texture	color	shape	texture	color
shape biased DS	86%	81%	74%	76%	77%	66%	64%	61%	60%
texture biased DS	52%	51%	41%	67%	73%	63%	52%	57%	54%
color biased DS	59%	56%	54%	56%	61%	59%	67%	75%	75%

C.4 The GAN model in Cross Feature Imagination

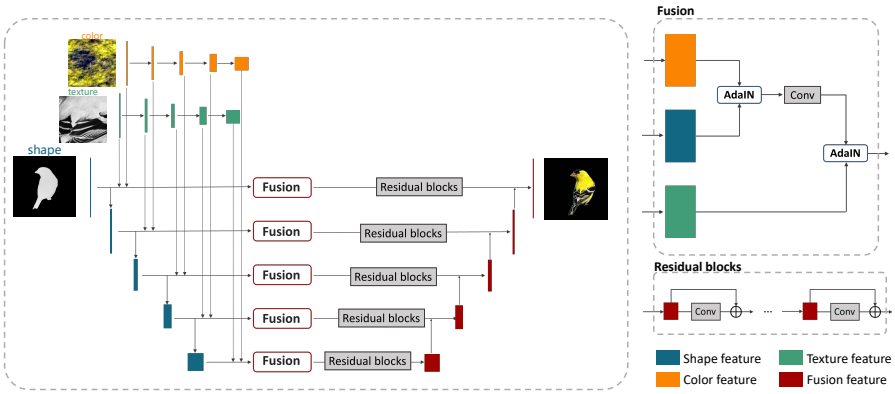

Fig. 8: The architecture of the GAN generator in cross feature imagination.

As introduced in main paper Sec. 5.2, we design a cross-feature pixel2pixel GAN model to generate the final image.

Fig. 8 shows the architecture of the generator. In order to ensure the quality of image generation, we set $k = 1$ when extracting the texture feature (the selection of k is introduced in Sec. A.2). Each feature, shape, texture, and color are encoded by a sequence of convolutions maps to $F_{s,i}, F_{t,i}, F_{c,i}, i = 1, 2, ..., K$. These features are fused into F_i by K feature fusion modules. As for fusion modules, we apply AdaIN and L residual blocks to blend the different features. The outputs are de-convoluted into $H_i, i = 1, 2, ..., K$, and then we get the final result. In our experiments, we use $K = 5$ and $L = 5$.

We use the same discriminator and loss function as [3]. In training the GAN model. We use Adam optimizer with $\beta_1 = 0.5$ and $\beta_2 = 0.999$ for both generator and discriminator, and set batch size to 16. The model is trained for 200 epochs on the training set, and we use the learning rate at $2e-4$ in the first 100 epochs and 2e-5 from epoch 100 to 200. To compare our result, we use the original pix2pix GANs [3] as our baseline model. The GAN model takes one type of feature as input and the original image as output. We trained three separate pix2pix GANs for each feature.

In Fig. 9, we show more results about the imagination on the test set when different types of features are given (denoted as sub-image a, b, c). With one feature (Line I), the retrieval model can find the other two plausible features (Line II), and then the GAN model can imagine the whole objects (Line III). Comparing the imagination results and the original images to which the input features belong (Line-IV), we can find that they have a similar input feature, while the other two features of the imagination images are decided by the retrieved features.

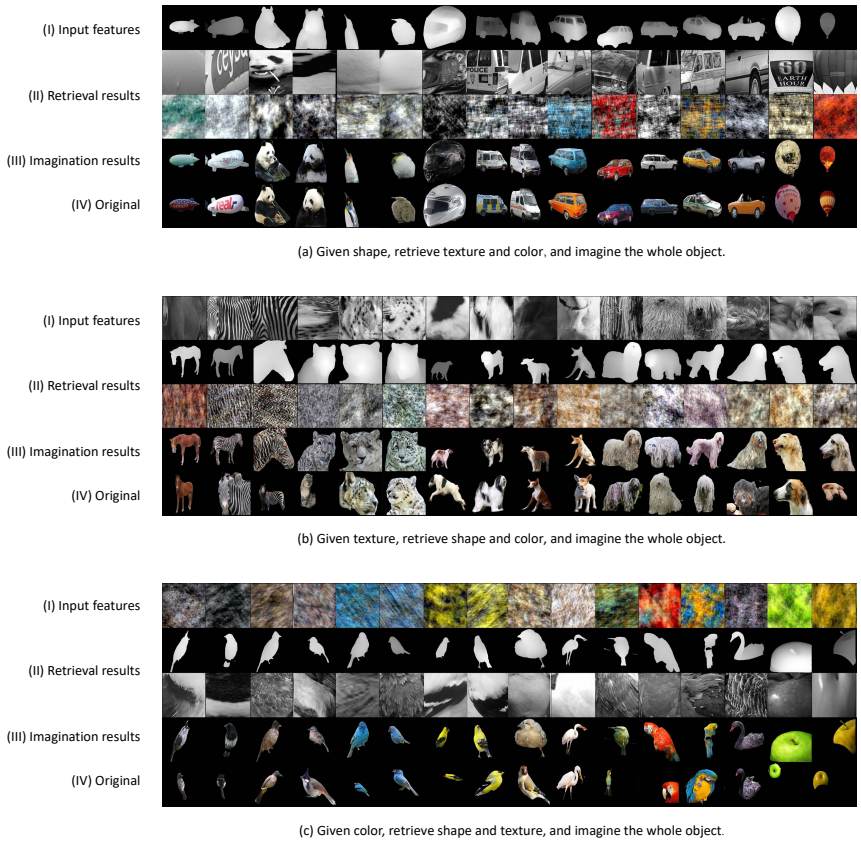

Fig. 9: Results of the imagination when different types of features are given (see sub-images a, b, c). Given one feature (Line I), the retrieval model can find the other two plausible features (Line II), and then the GAN model can imagine the whole objects (Line III). Line IV shows the original images of the input features.

D Discussion of Limitations and Future Work

As we use Grad-Cam as the foundation for selecting foreground parts, sometimes models could output the wrong saliency map (for example, the pre-trained models may regard water as the most important object for classifying a boat. As a result, our image parsing pipeline would take water as the foreground). So how to select accurate foreground parts is the first step we need to improve in the future.

After getting the foreground part of images, we use three feature extractors to mimic the process of humans perceiving this world. Our feature extractors are far away from perfect. In this work, we concatenate four square patches to get I_t, but

this introduced two lines in the I_t (the line between different squares). These lines may cause confusion to the neural networks. What's more, 4 patches may not be the best choice to represent the object's texture. Some classes are too complex to be represented by only 4 patches. For example, cars have windows, metal, rubber, paint, and plastic. Different parts of cars could have different textures. 4 square patches can represent the texture of cars to some degree, but if we want to get the representation of all texture information without any omission, we need to improve the work process of the texture extractors. However, the main goal of our model design is to summarize the contributions by end-to-end learning, with minimal human-introduced bias and assumptions in the architecture design. We mainly provide the first fully objective, data-driven, and indeed first-order, measure of the respective contributions.

In the open-world zero-shot learning experiment, we use a prototype dataset that contains limited classes, we will explore this direction with a larger dataset.

We think our HVE take a small but important step towards, from a humanoid perspective, better understanding the contributions of shape, texture, color to classification, zero-shot learning, imagination, and beyond.

Dataset copyright. We used publically available data. ImageNet, CUB, iLab-20M.

Imagenet license: Researcher shall use the Database only for non-commercial research and educational purposes We use a subset of the ILSVRC2012 dataset (ImageNet) and iLab-20M. The object in iLab-20M is toy vehicles, under Creative Commons CC-BY license. They do not contain any personally identifiable information offensive content.

References

1. Borji, A., Izadi, S., Itti, L.: ilab-20m: A large-scale controlled object dataset to investigate deep learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2221–2230 (2016)
2. Gao, M., Du, Y., Ai, H., Lao, S.: A hybrid approach to pedestrian clothing color attribute extraction. In: 2015 14th IAPR International Conference on Machine Vision Applications (MVA). pp. 81–84. IEEE (2015)
3. Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1125–1134 (2017)
4. Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D.: Grad-cam: Visual explanations from deep networks via gradient-based localization. In: Proceedings of the IEEE international conference on computer vision. pp. 618–626 (2017)
5. Snell, J., Swersky, K., Zemel, R.: Prototypical networks for few-shot learning. Advances in neural information processing systems **30** (2017)
6. Speer, R., Chin, J., Havasi, C.: Conceptnet 5.5: An open multilingual graph of general knowledge. In: Thirty-first AAAI conference on artificial intelligence (2017)
7. Van De Weijer, J., Schmid, C., Verbeek, J., Larlus, D.: Learning color names for real-world applications. IEEE Transactions on Image Processing **18**(7), 1512–1523 (2009)
8. Wah, C., Branson, S., Welinder, P., Perona, P., Belongie, S.: The caltech-ucsd birds-200-2011 dataset (2011)