Appendix for Training Interpretable Convolutional Neural Networks by Differentiating Class-specific Filters

Haoyu Liang*[1], Zhihao Ouyang*[2,4], Yuyuan Zeng[2,3], Hang Su†[1], Zihao He[5], Shu-Tao Xia[2,3], Jun Zhu†[1], and Bo Zhang[1]

[1]Dept. of Comp. Sci. and Tech., BNRist Center, Inst. for AI, THBI Lab, Tsinghua University, Beijing 100084, China [2]Tsinghua SIGS, Shenzhen 518055, China [3]Peng Cheng Lab [4]ByteDance AI Lab [5]Dept. of CS, University of Southern California
{lianghy18@mails, oyzh18@mails, zengyy19@mails, suhangss@mail, xiast@sz, dcszj@mail, dcszb@mail}.tsinghua.edu.cn zihaoh@usc.edu

A The Theoretical Convergence Interval for L1-density

This section derives the theoretical convergence interval for the L1-density of the CSG matrix $G \in [0,1]^{C\times K}$. L1-density is defined as $\frac{\|G\|_1}{CK}$. Now let's find the bound for the L1-density when a CSG CNN converges.

Lower bound In CSG training, we use projected gradient descent (PGD) to restrain G in the solution space $\{G \in [0,1]^{C\times K} | \left\|G^k\right\|_\infty = 1\}$. Therefore, for any G in the space, it is ensured that

$$\frac{\|G\|_1}{CK} = \frac{\sum_{k=1}^{K}\|G\|_1}{CK} \geq \frac{\sum_{k=1}^{K}\|G\|_\infty}{CK} = \frac{\sum_{k=1}^{K}1}{CK} = \frac{1}{C}.$$

Therefore, $\frac{1}{C}$ is lower bound for the L1-density of G, which also holds when the CSG CNN converges.

Upper bound In CSG training, we use $\lambda_2 d(\|G\|_1, g)$ as the sparsity regularization to punish $\|G\|_1$ when it is larger than g which is a hyperparameter as the upper bound for $\|G\|_1$. If we set λ_2 as a relatively large number, the sparsity regularization is strong enough to reduce $\|G\|_1$ under g before convergence. Therefore, we get the upper bound for the L1-density of G on convergence as

$$\frac{\|G\|_1}{CK} \leq \frac{g}{CK}.$$

Combining the lower bound and the upper bound above, the convergence interval for the L1-density of G is $[\frac{1}{C}, \frac{g}{CK}]$.

*Haoyu Liang and Zhihao Ouyang contributed equally.
†Hang Su and Jun Zhu are corresponding authors.

B Training setting and dataset preprocess

The ResNet20s are trained on CIFAR-10 [7]. The default settings include: batch size=256; SGD optimizer with momentum=0.9 [11]; initial learning rate=0.1; total training epochs=200; and every 1 in 3 epochs are in CSG path.

The ResNet18s are trained on ImageNet [3]. The default setting are the same as ResNet20s except trained on 4 gpus for 120 epochs .

The ResNet152s are finetuned on PASCAL VOC 2010 [4] from model pre-trained on ImageNet [3]. Parameters are frozen except the last 2 bottleneck blocks, gate matrix and linear layers. The first 10 epochs are trained in STD path, and after that every 2 in 3 epochs are in CSG path. The setting is: batch size=32; Adam optimizer [6]; initial learning rate=1e-5 for STD path, 1e-3 for CSG path; total training epochs=150.

We preprocess PASCAL VOC to be a classification dataset for training ResNet152s: we crop out images for the objects in 6 classes (bird, cat, dog, cow, horse and sheep) and resize the image to 128x128; then randomly reassign 3644 objects for training and 1700 objects for testing. No segmentation label is used in training. In testing phase, we only run the STD path which reuses the weights in the LSG path as shown in Fig 3.2.

The choice of backbone network are meticulously considered: 1) We finetune ImageNet models on a subset of PASCAL with quite few samples. We chose resnet152 to ensure the performance of baselines; 2) training large models on ImageNet from scratch is costly, while resnet18 is also a common choice on ImageNet; 3) We trained resnet20/50/101 on CIFAR10 and they consistently support our CSG conclusion. Since the penultimate layer in resnet20 has only 64 filters which ease visualization in Fig 4 and 7(a1,a2)

C Similarity Between Feature Vectors and Gates

To measure the directional similarity between the feature vector for class y and the gate vector for a class c, we design a similarity based on cosine.

For a pair of image and label (x, y) in the dataset D, input x into the CSG CNN, we get the average-pooled activation from the class-specific filters. Let's call it the feature vector for x and denote it as $a(x) \in \mathbb{R}^K$. Therefore the mean feature vector for class y on dataset D is

$$a_y(D) = \underset{(x,y)\in D}{\text{mean}}\, a(x).$$

Meanwhile, the gate vector for class c is G_c, the c-th row in the CSG matrix G.

Thus we can define the directional similarity between the feature vector for class y and the gate vector for a class c as

$$S_{yc}(D) = \cos(a_y(D), G_c).$$

In this way, we get a similarity matrix $S(D) \in [0, 1]^{C\times C}$ for the dataset D. If we take D as the set of all true positive samples and all false negative samples, we

can calculate the similarity matrices S_{TP}, S_{FN} respectively. Intuitively, a larger directional similarity $S_{yc}(D)$ means the feature vector is more closely related to the classes.

D Explanation for Filter Orthogonality

In this part, we give an intuitive explanation about why CSG training encourages filters for different classes become orthogonal (Paper Sec 4.3). Given a class c and a gate matrix that assigns the filter k for class c and filter k' for other class. During training, filter k' is blocked (i.e., its activation is masked) in the CSG path when class c's images input. In order to ensure the STD and CSG path generate similar outputs, the filter k' tends to be activated by class c as less as possible, which implies the weight of filter k' is approximately perpendicular to V_c (the linear space spanned by class c's features in a layer before). The filter k for class c, however, tends to be activated by class c as saliently as possible so as to enable the CNN to recognize this class. So the weight of filter k is approximately within V_c. Overall, the weights of filter k and filter k' tends to be orthogonal.

E Manually Fixed Gate Matrix

In Paper Sec 4.3 we manually initialize the gate matrix and fix it when training ResNet20 on CIFAR-10 from scratch. The gate matrix is visualized in Fig. E.1.

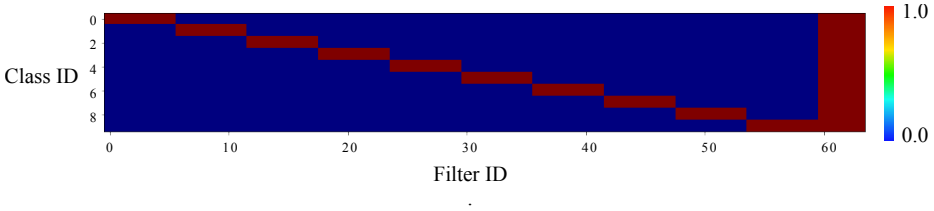

Fig. E.1. Manually fixed gate matrix for the ResNet on CIFAR10 trained in Paper Sec 4.3. Each class monopolizes 6 filters and 4 extra filters are shared by all classes.

We apply this setting based on a statistic analysis on 20 different CSG ResNet20s on CIFAR-10, which aims to figure out how many filters are monopolized by a class. The converged CSG matrices indicate that each class tends to monopolize about $m_1 = 6$ filters and the rest about $m_2 = 4$ filters are shared by classes. Following the statistic analysis, we tighten the constraint by manually setting a fixed CSG matrix for ResNet20s, where each class monopolizes 6 filters and 4 extra filters are shared by all classes. Similarly, we set $m_1 = 25$ and $m_2 = 6$ for AlexNets. They are the CSG matrices we use in Paper Sec 4.3.

F Cluster center experiments

Using the model with fixed gate matirx mentioned in Paper Sec 4.3, we train ResNet20s on CIFAR-10 with joint CSG and STD training. Then we run k-means clustering on the feature vectors after the global average pooling in the CSG/STD CNN. The clustering centers are visualized in Fig. F.2. We find that compared to the STD CNN, the CSG CNN yields better clustering centers, which form groups by channel that is almost the same as the gate matrix visualized in Fig. E.1.

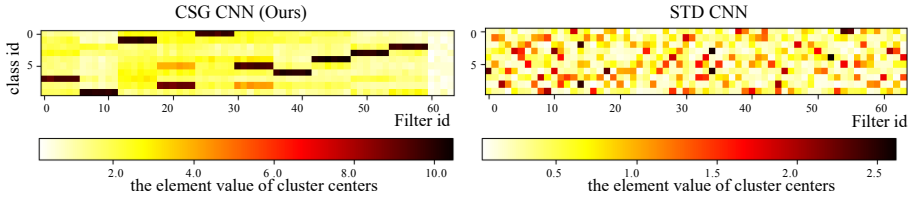

Fig. F.2. K-means cluster center of the feature vectors from ResNet20 train in Paper Sec 4.3. The x-axis is the channel id, and the y-axis is class id. Each row is a the mean of a cluster in the feature vectors' space and the color represents the value of an element in the mean.

G Localization Techniques

In Paper Sec 5.1, we study CSG CNNs' performs on localizing object classes with three localization the techniques based on filters, including gradient-based saliency map (GradMap) and activation map (ActivMap) for a single filter and classification activation map (CAM) for all filters.

To get a GradMap, we calculate the gradient of a filter's average-polled activation with respect to the input image. Then we normalized the gradient map with its second moment, apply gaussian blur (sigma=5 pixels) and segment out the region with values above 1.0.

To get an ActMap, we bilinear interpolate the activation map of a filter to input size and segment the region with values above the top 30% activation of the filter on the entire test dataset.

To get a CAM, we sums up all filters' activation maps with the weights of the linear connections between each channels and an output class [1]. By bilinear interpolating the sum activation map to input size and segment the region with the top 30% values in it, we get a classification activation map (CAM) [1], which is segmentation map for a class.

[1]CAM only works for CNNs ended with a global average pooling and one linear layer.

H Metrics for Localization

This section gives detailed definition of the metrics we use to evaluate our localization performance in Paper Sec 5.1.

H.1 Localization with One Filter

For an image x in class c (denoted as $x \in D_c \subset D$, where D is the dataset, D_c is the set of images with label c), we denote the ground-truth segmentation map for x as $S_x \in \{0,1\}^{H \times W}$, and denote the segmentation map given by filter k as $\hat{S}_x^k \in \{0,1\}^{H \times W}$. The $\hat{S}_x^k$ is calculated as $\hat{S}_x^k = \mathrm{I}\{\mathrm{resize}(A_k) \geq threshold\}$, which means resizing A_k (the activation map from filter k) to input size and then thresholding it.

The metrics for a filter on localization is defined below.

The IoU (intersection over union) for filter k on image x is defined as [2] [3]

$$\mathrm{IoU}_x^k := \frac{\left\| S_x \wedge \hat{S}_x^k \right\|_0}{\left\| S_x \vee \hat{S}_x^k \right\|_0}.$$

The Avg-IoU (average intersection over union) for filter k on localizing class c is defined as

$$\mathrm{IoU}_c^k := \operatorname*{mean}_{x \in D_c} \mathrm{IoU}_x^k.$$

The APn (average precision $n\%$) for filter k on localizing class c is defined as

$$\mathrm{APn}_c^k := \operatorname*{mean}_{x \in D_c} \mathrm{I}\{\mathrm{IoU}_x^k \geq n\%\}.$$

When $c^* = \arg\max_c \mathrm{APn}_c^k$, we call filter k is focused on localizing class c^*, denoted as $k \in F_{c^*}$, where F_{c^*} is the set of filters focused on localizing class c. Therefore the localization performance for filter k can be evaluated with $\mathrm{IoU}^k := \mathrm{IoU}_{c^*}^k$ and $\mathrm{APn}^k := \mathrm{APn}_{c^*}^k$.

The metrics averaged for all filters on localization is defined below based on the aforementioned metrics.

(1) The Avg-IoU and APn for localizing class c as

$$\mathrm{IoU}_c := \mathrm{mean}_{k \in F_c} \mathrm{IoU}^k,$$

and

$$\mathrm{APn}_c := \mathrm{mean}_{k \in F_c} \mathrm{APn}^k.$$

(2) The Avg-IoU and APn for localizing all classes is defined as

$$\mathrm{IoU} := \mathrm{mean}_{k \in \{1,2,\ldots,K\}} \mathrm{IoU}^k,$$

and

$$\mathrm{APn} := \mathrm{mean}_{k \in \{1,2,\ldots,K\}} \mathrm{APn}^k.$$

[2] For $A, B \in \{0,1\}^{H \times W}$,
define $A \vee B^{H \times W}$ as $(A \vee B)_{ij} = \max(A_{ij}, B_{ij})$;
define $A \wedge B^{H \times W}$ as $(A \wedge B)_{ij} = \min(A_{ij}, B_{ij})$.

[3] $\|\cdot\|_0$ is the number of non-zero elements.

H.2 Localization with All Filters

For an image x in class c (denoted as $x \in D_c$), we denote the ground-truth segmentation map for x as $S_x \in \{0,1\}^{H \times W}$, and denote the segmentation map given by the classification activation map (CAM) [1] as $\hat{S}_x \in \{0,1\}^{H \times W}$. The $\hat{S}_x$ is calculated as $\hat{S}_x = \mathrm{I}\{\mathrm{resize}(\sum_k W_c^k A_k) \geq threshold\}$, where A_k is the activation map of filter k, and W_c^k is the weight of the linear connection between filter k and the logit for class c.

The IoU (intersection over union) for CAM on image x is defined as

$$\mathrm{IoU}_x := \frac{\left\| S_x \wedge \hat{S}_x \right\|_0}{\left\| S_x \vee \hat{S}_x \right\|_0}.$$

(1) The metrics for localizing a class is defined below.

The Avg-IoU (average intersection over union) for localizing class c is defined as

$$\mathrm{IoU}_c := \mathrm{mean}_{x \in D_c} \mathrm{IoU}_x.$$

The APn (average precision $n\%$) for localizing class c is defined as

$$\mathrm{APn}_c := \mathrm{mean}_{x \in D_c} \mathrm{I}\{\mathrm{IoU}_x \geq n\%\}.$$

(2) The metrics for localizing all classes is defined below. The Avg-IoU for localizing all classes is defined as

$$\mathrm{IoU} := \mathrm{mean}_{x \in D} \mathrm{IoU}_x,$$

The APn for localizing all classes is defined as

$$\mathrm{APn} := \mathrm{mean}_{x \in D} \mathrm{I}\{\mathrm{IoU}_x \geq n\%\}.$$

I Detailed Settings in Adversarial Sample Detection

In Paper Sec 5.2 we generate the non-targeted adversarial samples with commonly used white-box attack. The setting for them is: FGDM [5] ($\epsilon = 0.031$), PGD [8] ($\epsilon = 0.031$, $iter = 7$) and CW [2] ($max\_iterations = 100$). The adversarial target classes are from a random permutation of original classes besides each image's true class. We randomly select 500, 1000, 2000 images per class in CIFAR-10 to form different sizes of training datasets and 100 images per class for the testing.

J Defending Adversarial Samples

In this experiments, inspired by using class-specific filters to detect adversarial samples, we further explore CSG CNNs' potential in defending adversarial attacks. We use the models (CSG/STD ResNet20) and the dataset (CIFAR10) the

Table J.1. Black Box Attack on STD CNN and CSG CNN

Attack	Metric	STD CNN	CSG CNN
No Attack	Accuracy	88.03%	88.85%
Single Pixel Attack	Attack Success Rates	14.00%	2.00%
Local Search Attack		15.00%	2.00%

same as Paper Sec 5.2. Two black box attacks are conducted, including one pixel attack [10] and local search attack [9]. They try to fool models according to the model's predicted probability without access to the models' parameters and architectures. From the results shown in Table J.1, we find both the attacks gain attack success rates on the CSG CNN much lower than on the STD CNN. This demonstrates that CSG training also improves robustness of CNNs in defending adversarial attacks. We guess the robustness is caused by the increase of within-class distance and the decrease of between-class distance, which requires further verification yet. Robustness on defending adversarial attacks is another valuable characteristic of the highly class-related representation from our class-specific filters.

References

1. Bau, D., Zhou, B., Khosla, A., Oliva, A., Torralba, A.: Network dissection: Quantifying interpretability of deep visual representations. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 6541–6549 (2017)
2. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: S&P (2017)
3. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2009)
4. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Results. http://www.pascal-network.org/challenges/VOC/voc2010/workshop/index.html
5. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: International Conference on Learning Representations (2014)
6. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization (2014), http://arxiv.org/abs/1412.6980
7. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. Technical Report TR-2009 (2009)
8. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083 (2017)
9. Narodytska, N., Kasiviswanathan, S.P.: Simple black-box adversarial perturbations for deep networks. arXiv preprint arXiv:1612.06299 (2016)
10. Su, J., Vargas, D.V., Sakurai, K.: One pixel attack for fooling deep neural networks. IEEE Transactions on Evolutionary Computation (2019)
11. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: International Conference on Machine Learning. pp. 1139–1147 (2013)