Improving Model Robustness via Automatically Incorporating Self-supervision Tasks

Donghwa Kim
KAIST
donghwa.kim@kaist.ac.kr

Kangwook Lee
UW-Madison
kangwook.lee@wisc.edu

Changho Suh
KAIST
chsuh@kaist.ac.kr

Abstract

Robust deep learning models are of great practical importance. Hendrycks et al. [12] have recently shown that one can train a robust deep learning model by incorporating a self-supervision loss function while training. In this work, we show that one can further improve the robustness by delicately incorporating multiple self-supervision tasks with the aid of AutoML. To this end, we first study a variety of self-supervision tasks and show that each of them can be used to improve the robustness of a trained model. We then propose an AutoML-based algorithm that automatically strikes the right balance between multiple self-supervision terms. Our experiments show that our AutoML algorithm, together with multiple self-supervision tasks, can achieve the best test accuracy on CIFAR-10-C, a benchmark for robust image classification. The code is available at https://github.com/storykim/robustness-self-supervision.

1 Introduction

While self-supervision is known to help learn representation with scarce labeled data [8, 5], it was unclear whether it could further benefit supervised learning with a sufficiently large labeled data. In a recent work by Hendrycks et al. [12], the authors show that one can improve the model robustness by incorporating the self-supervision loss. More specifically, consider a scenario where one wants to train a classifier. The main loss can be defined as the cross-entropy between the output probabilities of the neural network and the labels, say $\mathcal{L}_{\text{main}}(X, Y; \theta_{\text{c}}, \theta_{\text{main}})$, where (X, Y) is a labeled dataset, θ_{c} is the parameter corresponding to the (common) lower part of a neural network, and θ_{main} is the parameter corresponding to the (task-specific) upper part of a neural network. Instead of minimizing this loss term alone, the authors propose the use of an auxiliary task inspired by self-supervision. More specifically, one simultaneously solves a self-supervision task with another neural network that shares the lower part of the neural network with the main neural network. Without loss of generality, the loss of a self-supervision task can be written as $\mathcal{L}_{\text{aux}}(X; \theta_{\text{c}}, \theta_{\text{aux}})$. Note that it does not involve Y and involves the common parameter θ with the main loss term. Given these loss functions, they train a model by solving the following optimization problem

$$\min_{\theta, \theta_{\text{main}}, \theta_{\text{aux}}} \mathcal{L}_{\text{main}}(X, Y; \theta_{\text{c}}, \theta_{\text{main}}) + \lambda \mathcal{L}_{\text{aux}}(X; \theta_{\text{c}}, \theta_{\text{aux}}), \tag{1}$$

where λ is a hyperparameter that determines the ratio between the two loss terms.

Since the authors of [12] mostly focus on the use of a single self-supervision task, it is not clear whether or not one can further improve the robustness of a model by incorporating multiple self-supervision tasks. This precisely sets the goal of our work. We first identify several self-supervision tasks, each of which can improve the robustness of a model when used in conjunction with the main loss. We then show one can incorporate multiple self-supervision tasks at the same time to further improve the robustness of a model. To achieve this goal, we propose an AutoML-based algorithm that automatically tunes the weights associated with different auxiliary tasks. Our experimental results show that our AutoML-based approach can achieve superior robustness over the existing approaches.

2 Related Work

Meta learning and AutoML: Meta-learning has made rapid progress in the past years. The goal of meta-learning is learning to learn better based on data, and it can be used for systematically improving the performance of transfer learning [6, 15], optimization/learning algorithms [1, 2, 17, 4], deep learning models [22, 16, 18], data preprocessing/ augmentation [19, 3], and reinforcement learning [7, 9, 20].

Self-supervision: Self-supervision is a technique that creates labels from unlabeled data and leverages those self-labeled data to learn data representation. To be more concrete, consider an image dataset without labels. One can generate a labeled dataset by manipulating these unlabeled images. For instance, one can rotate an image by an angle $\in \{0°, 90°, 180°, 270°\}$ and assign the image an angle label [8]. Another representative example is autoencoders [13]. Here, an unlabeled example X_i is labeled by itself, giving us a labeled example (X_i, X_i). A variety of self-supervision tasks have been proposed in the literature, and self-supervision has been shown effective, particularly when labeled data is scarce. While self-supervision has been known useful for learning representation with scarce labeled data, it was not clear whether it has any value when one is equipped with a sufficient amount of labeled data. Recently, Hendrycks et al. [12] show that self-supervision can be useful even in this case as it can help us improve the robustness of deep learning models.

3 Single Self-supervision Task for Model Robustness

In this section, we show that the algorithm proposed in [12] indeed works not only with the rotation task but also with various self-supervision tasks. Specifically, we examine the following seven self-supervision tasks. See Fig. 1 for a visual illustration of the following self-supervision tasks.

- Rotation: Each input image is rotated by an angle $\in \{0°, 90°, 180°, 270°\}$. The goal of this self-supervision task is to correctly predict the rotation angle given a rotated image. The loss is defined as the cross-entropy between the ground truth label and the prediction results.
- Coloring (Color): Each input images is converted into a grayscale image. The goal is to correctly recover the color of each pixel. Here, we use the CIELAB representation of colors [21], in which each color is represented by a three-dimensional vector (L, A, B). Roughly speaking, L denotes the lightness, A denotes redness, and B denotes yellowness. The CIELAB representation is known to better capture human visual perception. We drop the L value and quantize the values of A and B into 32 bins. We then define the loss function for each pixel as the sum of two cross-entropy terms, one for predicting A and the other for predicting B. Thus, the total loss is $\frac{1}{2N_{\text{pixel}}} \sum_{\text{pixel}} \sum_{\text{channel} \in \{A,B\}} CE(Y_{\text{pixel,channel}}, \hat{Y}_{\text{pixel,channel}})$, where CE is a shorthand term of cross-entropy, N_{pixel} is the number of pixels in an image, and $Y, \hat{Y} \in \{1, 2, \ldots, 32\}$.
- Image reconstruction (IR): The goal is to correctly reconstruct the input image. The loss is defined as the mean squared error over pixels. Note that this task is identical to autoencoders.
- Image deblurring (BIR): Same as above except that the input image is blurred. We call this task BIR (blurred-image reconstruction) as it can be viewed as reconstructing blurred images.
- Inpainting (Inpaint): Same as above except that a central part of the input image is removed.
- Color-channel guessing (RGB guess): Each image's RGB channels are randomly shuffled. The goal is to identify the original color-shuffling pattern out of $3! = 6$ possible ones. The loss is defined as $CE(Y, \hat{Y})$, where $Y, \hat{Y} \in \{1, 2, \ldots, 6\}$.
- Jigsaw puzzle (Jigsaw): Each image is split into two by two pieces, and the four pieces are randomly reordered. The goal is to identify the shuffling pattern out of $4! = 24$ possible ones. The loss is defined as $CE(Y, \hat{Y})$, where $Y, \hat{Y} \in \{1, 2, \ldots, 24\}$.

Each self-supervision task is solved by a neural network whose lower part is shared with the main neural network and whose upper part is designed specifically for the corresponding task. See the supplemental materials for more details about the network designs. We run the training algorithm proposed in [12] in conjunction with each of the above self-supervision tasks and evaluate its robustness. We minimize the overall loss function using SGD with momentum with the momentum parameter of 0.9 on CIFAR-10 [14]. We also apply the weight decay with the decay factor of 0.0005.

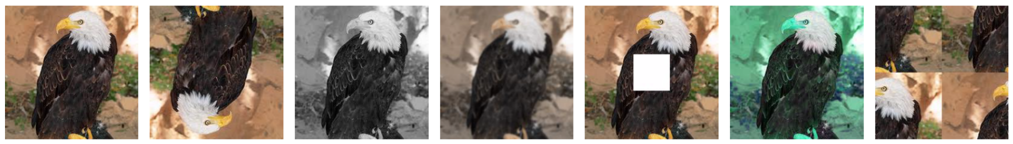


Figure 1: Inputs generated for self-supervision tasks. From left to right: Original, rotation, coloring, image deblurring, inpainting, color-channel guessing, jigsaw puzzle.

For learning rates, we use the cosine annealing with the initial value of 0.1 and the minimum value of 10^{-6}. The hyperparameter λ is fixed as 0.5 for all tasks. We use minibatches of size 64 to estimate the main loss/gradients. To measure the robustness of a model, we use CIFAR-10-C dataset [11]. The CIFAR-10-C dataset contains 950,000 images which are algorithmically generated with 19 kinds of corruption types and five levels of severity based on CIFAR-10 validation set. Summarized in Table 1 are our experimental results. One can see that all of the tested self-supervision tasks except the last one helps us obtain a more robust model.

Table 1: Test accuracy on CIFAR-10-C

	Baseline	Rotation	Color	IR	BIR	Inpaint	RGB guess	Jigsaw
Acc.	74.07%	76.37%	75.06%	74.73%	75.84%	74.69%	74.15%	73.41%

4 An AutoML Algorithm for Incorporating Multiple Self-supervision Tasks

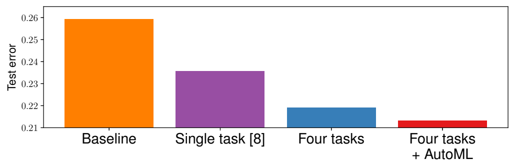
A natural question that arises is whether one can incorporate multiple self-supervision tasks to further improve the robustness of the model. That is, one would like to incorporate k self-supervision tasks and obtain a more robust model than those trained with a single self-supervision task. This can be achieved by training a model with the following loss function $\mathcal{L}_{\text{main}}(X, Y; \theta_{\text{c}}, \theta_{\text{main}}) + \sum_{i=1}^{k} \lambda_i \mathcal{L}_{\text{aux},i}(X; \theta_{\text{c}}, \theta_{\text{aux},i})$, where $\mathcal{L}_{\text{aux},i}$ is the ith auxiliary loss function, $\theta_{\text{aux},i}$ is the parameter of each task-specific neural network, and λ_i is the hyperparameter that determines the weight of the corresponding term. It is clear that one must carefully choose the value of λ_i's in order to obtain a robust model. For instance, if the values of λ_i's are too low, the training algorithm reduces to the standard supervised training, and hence one cannot expect the model to perform well under robustness settings. On the other hand, if these values are chosen too high, the main loss function will be ignored, making the main model perform poorly on the main prediction task. Therefore, it is critical of importance to carefully optimize these hyperparameters. To this end, we propose an AutoML algorithm that can automatically adjust these hyperparameters, striking a balance between multiple self-supervision tasks.

Recall that the goal is to minimize the main task loss in conjunction with multiple self-supervision tasks. Denote the concatenation of all model parameters by ϕ, i.e., $\phi = (\theta_{\text{c}}, \theta_{\text{main}}, \theta_{\text{aux},1}, \dots, \theta_{\text{aux},k})$. Similarly, denote the concatenation of the model parameters associated with the main task by ψ, i.e., $\psi = (\theta_{\text{c}}, \theta_{\text{main}})$. To automatically adjust the values of λ_i's, we define the following metaloss.

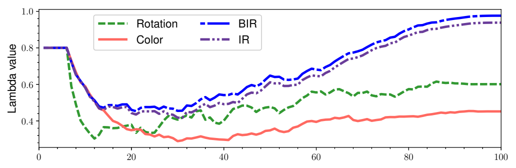
$$\mathcal{L}_{\text{meta}} := \mathcal{L}_{\text{main}} \left(X_{\text{val}}, Y_{\text{val}}; \psi - \alpha \frac{\partial \mathcal{L}_{\lambda_1, \lambda_2, \dots, \lambda_k}}{\partial \psi} \right). \tag{2}$$

That is, the meta loss captures the performance of the updated model parameter of the main network $(\psi - \alpha \frac{\partial \mathcal{L}_{\lambda_1, \lambda_2, \dots, \lambda_k}}{\partial \psi})$ on the validation data. Note that $\mathcal{L}_{\text{meta}}$ is a function of $\{\lambda\}$.

With these definitions, one can apply the following AutoML algorithm to adjust the values of $\{\lambda\}$ while training the main model. In each iteration, we first compute $\mathcal{L}_{\lambda_1, \lambda_2, \dots, \lambda_k}$ and then backpropagate it with respect to ψ, obtaining $\frac{\partial \mathcal{L}_{\lambda_1, \lambda_2, \dots, \lambda_k}}{\partial \psi}$. From this, one can compute the updated model parameter $\psi - \alpha \frac{\partial \mathcal{L}_{\lambda_1, \lambda_2, \dots, \lambda_k}}{\partial \psi}$. Then, one can compute the validation loss (of the main task) of this updated model parameter, obtaining $\mathcal{L}_{\text{meta}}$. By differentiating $\mathcal{L}_{\text{meta}}$ with respect $\{\lambda\}$, we can calculate $\frac{\partial \mathcal{L}_{\text{meta}}}{\partial \lambda_1}, \frac{\partial \mathcal{L}_{\text{meta}}}{\partial \lambda_2}, \dots, \frac{\partial \mathcal{L}_{\text{meta}}}{\partial \lambda_k}$. Using these, we accordingly update the values of λ_i's, i.e., $\lambda_i \leftarrow \lambda_i - \beta \frac{\partial \mathcal{L}_{\text{meta}}}{\partial \lambda_i}$ for all $i \in \{1, 2, \dots, k\}$, where β is the learning rate for meta updates. Once we have updated the λ_i's, we now compute the gradient of the original loss function with respect to



(a) Test error on CIFAR-10-C



(b) The evolution of λ's

Figure 2: Experimental results with multiple self-supervision tasks + AutoML

all model parameters ϕ using the updated values of λ's and then update all the model parameters. While we assumed the simple stochastic gradient descent when computing the meta loss, one may actually update the main model with other algorithms such as Adam. To further stabilize the training procedure, we train the model with fixed λ's without learning meta updates for the first τ epochs, which we call the warm-up period. Our algorithm is summarized in Algorithm 1.

Algorithm 1 AutoML algorithm for incorporating multiple self-supervision tasks

```
**Input:** Initial model parameter $\phi$, LR schedule $\alpha_t$, meta LR $\beta$, warm-up period $\tau$
**repeat**
    $\mathcal{L}_{\lambda_1,\lambda_2,\dots,\lambda_k} \leftarrow \mathcal{L}_{\text{main}}(X_{\text{train}}, Y_{\text{train}}; \phi) + \sum_{i=1}^{k} \lambda_i \mathcal{L}_{\text{aux},i}(X_{\text{train}}; \theta_{\text{c}}, \theta_{\text{aux},i})$
    **if** epoch count $> \tau$ **then**
        Compute $\frac{\partial \mathcal{L}_{\lambda_1,\lambda_2,\dots,\lambda_k}}{\partial \psi}$ via backpropagation
        $\mathcal{L}_{\text{meta}} \leftarrow \mathcal{L}_{\text{main}}\left(X_{\text{val}}, Y_{\text{val}}; \psi - \alpha_t \frac{\partial \mathcal{L}_{\lambda_1,\lambda_2,\dots,\lambda_k}}{\partial \psi}\right)$    {Compute meta loss}
        Compute $\frac{\partial \mathcal{L}_{\text{meta}}}{\partial \lambda_1}, \frac{\partial \mathcal{L}_{\text{meta}}}{\partial \lambda_2}, \dots, \frac{\partial \mathcal{L}_{\text{meta}}}{\partial \lambda_k}$ via backpropagation    {Compute meta gradients}
        $\lambda_i \leftarrow \lambda_i - \beta \frac{\partial \mathcal{L}_{\text{meta}}}{\partial \lambda_i}$ for all $i \in \{1, 2, \dots, K\}$    {Update $\lambda$'s}
        $\mathcal{L}_{\lambda_1,\lambda_2,\dots,\lambda_k} \leftarrow \mathcal{L}_{\text{main}}(X_{\text{train}}, Y_{\text{train}}; \phi) + \sum_{i=1}^{k} \lambda_i \mathcal{L}_{\text{aux},i}(X_{\text{train}}; \theta_{\text{c}}, \theta_{\text{aux},i})$    {With updated $\lambda$'s}
    **end if**
    Compute $\frac{\partial \mathcal{L}_{\lambda_1,\lambda_2,\dots,\lambda_k}}{\partial \phi}$ via backpropagation
    $\phi \leftarrow \phi - \alpha_t \cdot \text{SGD}\left(\phi, \frac{\partial \mathcal{L}_{\lambda_1,\lambda_2,\dots,\lambda_k}}{\partial \phi}\right)$
**until** $\phi$ converges
```

We now apply our approach with the self-supervision tasks identified in the previous section. Here, we run our AutoML algorithm with the four most effective self-supervision tasks–rotation, image deblurring, coloring, and image reconstruction–and compare its performance with that without AutoML. Most of the experimental settings remain the same. We use the warm-up period of $\tau = 5$ epochs and use minibatches of size 25 to estimate the meta loss/gradients. For the validation dataset, we use 1% of the CIFAR-10-C dataset, consisting of 100 random samples from each category.

Shown in Figure 2a are our experimental results. The baseline algorithm that does not involve any self-supervision tasks achieves the test error of 25.93%, and the model trained with a single self-supervision task (rotation) gives us the error rate of 23.63%. By incorporating four self-supervision tasks together with AutoML, the test error becomes 21.32%, reducing the error rate of the existing approach by 9.7%. Figure 2b show how the values of λ's change during the training procedure. During the warm-up period, the values of λ's remain the same, and after the warm-up period, they start evolving according to the meta updates.

5 Conclusion

In this work, we first show that there exist diverse self-supervision tasks, each of which can improve the robustness of a deep learning model. Based on this observation, we propose an AutoML-based algorithm which can automatically balance the weights of multiple self-supervision losses, achieving the state-of-the-art robustness performance on CIFAR-10-C.

References

[1] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *NeurIPS*, 2016.

[2] Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Neural optimizer search with reinforcement learning. In *PMLR*, 2017.

[3] Ekin Dogus Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data. In *CVPR*, 2019.

[4] Ilias Diakonikolas, Gautam Kamath, Daniel Kane, Jerry Li, Jacob Steinhardt, and Alistair Stewart. Sever: A robust meta-algorithm for stochastic optimization. In *ICML*, 2019.

[5] Alexey Dosovitskiy, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with convolutional neural networks. In *NeurIPS*, 2014.

[6] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *JMLR*, 2017.

[7] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *PMLR*, 2018.

[8] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. In *ICLR*, 2018.

[9] Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. In *NeurIPS*. 2018.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[11] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *ICLR*, 2019.

[12] Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, and Dawn Song. Using self-supervised learning can improve model robustness and uncertainty. In *NeurIPS*, 2019.

[13] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 2006.

[14] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[15] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy Hospedales. Learning to generalize: Meta-learning for domain generalization. In *AAAI*, 2018.

[16] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018.

[17] Luke Metz, Niru Maheswaranathan, Jonathon Shlens, Jascha Sohl-Dickstein, and Ekin D. Cubuk. Using learned optimizers to make models robust to input noise. In *ICML Workshop on Uncertainty and Robustness in Deep Learning*, 2019.

[18] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *PMLR*, 2018.

[19] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. In *ICML*, 2018.

[20] Zhongwen Xu, Hado P van Hasselt, and David Silver. Meta-gradient reinforcement learning. In *NeurIPS*. 2018.

[21] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *ECCV*, 2016.

[22] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

A Supplemental Materials

A.1 Neural network architectures

Here, we provide the details of the neural network architectures used in our experiments.

- Common part (θ_{c}): 40-2 WideResNet [10]. The size of input image is $(32 \times 32 \times 3)$, and that of the output is (128).
- Main task (θ_{main}): $\mathrm{FC}(128, 10)$
- Rotation ($\theta_{\mathrm{aux,rot}}$): $\mathrm{FC}(128, 4)$
- Coloring (Color) ($\theta_{\mathrm{aux,color}}$): TransposedConv(channels=256, kernel=4, stride=1, padding=0) – BatchNorm – ReLU – TransposedConv(channels=128, kernel=4, stride=2, padding=1) – BatchNorm – ReLU – TransposedConv(channel=64, kernel=4, stride=2, padding=1) – BatchNorm – ReLU – TransposedConv(channel=64, kernel=4, stride=2, padding=1). The output tensor is of size $(32 \times 32 \times (32 \cdot 2))$. Each tensor of size $(1 \times 1 \times (32 \cdot 2))$ is viewed as the concatenation of 32-way classification results for A value and that for B value for a certain pixel. See Figure 3 for a visual illustration.
- Image reconstruction (IR) ($\theta_{\mathrm{aux,IR}}$): TransposedConv(channels=256, kernel=4, stride=1, padding=0) – BatchNorm – ReLU – TransposedConv(channels=128, kernel=4, stride=2, padding=1) – BatchNorm – ReLU – TransposedConv(channel=64, kernel=4, stride=2, padding=1) – BatchNorm – ReLU – TransposedConv(channel=3, kernel=4, stride=2, padding=1). The output tensor is of size $(32 \times 32 \times 3)$, corresponding to the reconstructed image.
- Image deblurring (BIR) ($\theta_{\mathrm{aux,BIR}}$): Same as above.
- Inpainting (Inpaint) ($\theta_{\mathrm{aux,inpaint}}$): TransposedConv(channels=64, kernel=4, stride=1, padding=0) – BatchNorm – ReLU – TransposedConv(channels=3, kernel=4, stride=2, padding=1) – Sigmoid. The output tensor is of size $(8 \times 8 \times 3)$, corresponding to the inpainting result.
- Color-channel guessing (RGB guess) ($\theta_{\mathrm{aux,RGB}}$): $\mathrm{FC}(128, 6)$
- Jigsaw puzzle (Jigsaw) ($\theta_{\mathrm{aux,jigsaw}}$): $\mathrm{FC}(128, 24)$

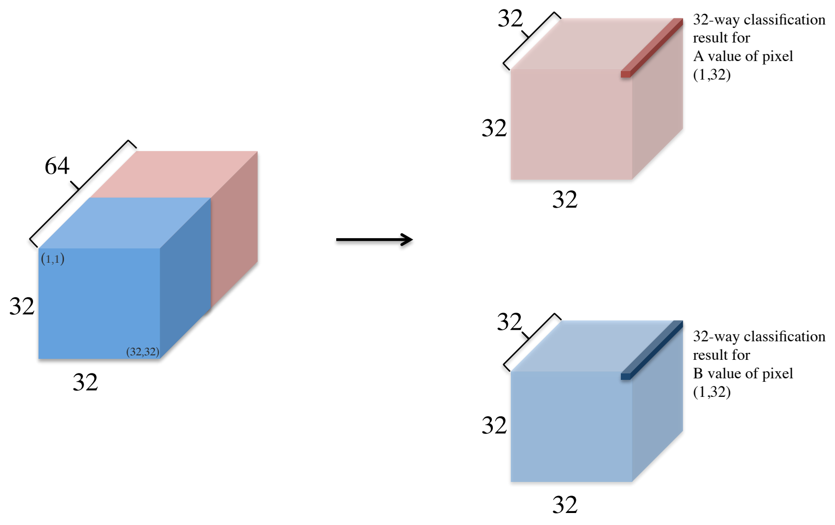

64
(1,1)
32
(32,32)
32
32
32-way classification result for A value of pixel (1,32)
32
32
32
32-way classification result for B value of pixel (1,32)
32
32

Figure 3: A visual illustration of $\theta_{\mathrm{aux,color}}$.