OPT2025: 17th Annual Workshop on Optimization for Machine Learning 1–12

Efficient Training of CNN Ensembles via Feature-Prioritized Boosting

Biyi Fang[1] biyifang2021@u.northwestern.edu
Truong Vo[1] truongvo2025@u.northwestern.edu
Jean Utke[2] jutke@allstate.com
Diego Klabjan[1] d-klabjan@northwestern.edu

[1]Northwestern University [2]Allstate

Abstract

Convolutional Neural Networks (CNNs) have achieved remarkable success in computer vision, yet training deep architectures with millions of parameters remains computationally expensive and design-intensive. We present a novel framework for efficient optimization of CNN ensembles that integrates subgrid-boosted feature selection with boosting-inspired learning. Our method introduces subgrid selection and importance sampling to emphasize statistically informative regions of the feature space, while embedding boosting weights directly into the ensemble training process through a least squares formulation. This design accelerates convergence, reduces the burden of manual architecture tuning, and enhances predictive performance. Across multiple fine-grained image classification benchmarks, our subgrid-boosted CNN ensembles consistently outperform conventional CNNs in both accuracy and training efficiency, demonstrating the effectiveness and generality of the proposed approach.

1. Introduction

In parallel, ensemble methods such as boosting have gained significant attention for their theoretical and empirical advantages over single-model approaches, especially in structured tasks like decision trees Quinlan (2004). To extend these benefits to CNNs, BoostCNN Moghimi et al. (2016) proposed combining shallow CNNs within a boosting framework, thereby simplifying the architecture design problem. However, this method suffers from high memory usage and long runtimes when the weak learners become moderately complex.

To overcome these limitations, we propose Subgrid BoostCNN, a novel boosting algorithms that builds upon BoostCNN by integrating ideas from feature subsetting in random forests. Each weak learner is trained on a dynamically selected subset of image pixels (i.e., a subgrid), reducing computational burden while maintaining representational fidelity. Our subgrid selection is guided by the image gradient and the residual from the boosting iteration, allowing the model to focus on the most informative regions of the image. This approach reduces training complexity by breaking the full spatial pixel dependencies, albeit with the potential trade-off of increased noise. Additionally, we propose a strategy to further reduce computational cost by avoiding full CNN re-optimization in each boosting iteration. Specifically, we reuse the convolutional layers from the previous learner and pair them with the fixed fully connected classifier from the initial iteration to compute pixel importance. This reused architecture enables us to train new learners efficiently with fewer parameters-requiring only one forward-backward pass per iteration, similar to BoostCNN but with significantly reduced overhead.

2. Algorithms

We summarize BoostCNN Moghimi et al. (2016) in Appendix B and propose Subgrid BoostCNN, which combines BoostCNN with a subgrid strategy. Training BoostCNN on full-size images with complex CNNs as weak learners is both time and memory intensive. A naïve solution is to downsize images directly, but this risks weakening strong signals and amplifying noise across learners. Another approach is to Random pixel sampling, but this method suffers from high variance, especially on sharp or noisy images. To address these issues, we apply the subgrid trick, where each weak learner operates on images defined by deleting subsets of rows and columns. This leads to variable input sizes across iterations, preventing parameter reuse between learners. To handle this, we separate each CNN into a feature extractor (all convolutional/pooling layers) and a classifier (final fully connected layers). We denote the base weak learner by g_0 and subsequent learners by g_t. Subgrid BoostCNN then assigns an importance index to each pixel (j, k), guiding subgrid selection across learners.

$$I_{j,k} = \frac{1}{|\mathcal{D}|} \sum_{(x_i,z_i)\in\mathcal{D}} \sum_{c\in C} \left| \frac{\partial \mathcal{L}(w, g)}{\partial x_i^{j,k,c}} \right|, \tag{1}$$

where $x_i^{j,k,c}$ denotes pixel (j, k) in channel c from sample i and C represents the set of all channels. The importance index of a row, column is a summation of the importance indexes in the row, column divided by the number of columns, rows, respectively. This importance index is computed based on the residual of the current predictor. Therefore, a larger importance value means a larger adjustment is needed for this pixel at the current iterate. The algorithm uses the importance index generated based on the feature extractor of the incumbent weak learner and the classifier from g_0 to conduct subgrid selection. The selection strategy we apply in the algorithm is deleting less important columns and rows, which eventually provides the important subgrid. After the subgrid is selected, subgrid BoostCNN creates a new tensor x_i^t at iterate t, and then feeds it into an appropriate feature extractor followed by a proper classifier. The modified minimization problem becomes

$$\mathcal{L}(w, g) = \sum_{(x_i,z_i)\in\mathcal{D}} \left\| g(x_i^t) - w(x_i, z_i) \right\|^2, \tag{2}$$

where the modified boosting classifier is

$$f(x) = \sum_{t=1}^{N} \alpha_t g_t(x^t). \tag{3}$$

In this way, subgrid BoostCNN dynamically selects important subgrids based on the updated residuals. Moreover, subgrid BoostCNN is able to deal with inputs of different sizes by applying different classifiers. Furthermore, we are allowed to pass the feature extractor's parameters from the previous weak learner since the feature extractor is not restricted to the input size. The proposed algorithm (subgrid BoostCNN) is summarized in Algorithm 1.

Algorithm 1 subgrid BoostCNN

```
 1: Inputs:
        number of classes $M$, number of boosting iterations $N_b$, shrinkage
        parameter $\nu$, dataset $\mathcal{D} = \{(x_1, z_1), \cdots, (x_n, z_n)\}$ where
        $z_i \in \{1, \cdots, M\}$ is the label of sample $x_i$, and $0 < \sigma < 1$
 2: Initialize:
        set $f(x) = \mathbf{0} \in \mathbb{R}^M$, $P_0 = \{(j, k) | (j, k) \text{ is a pixel in } x_i\}$
 3: compute $w(x_i, z_i)$ for all $(x_i, z_i)$, using (6)
 4: train a deep CNN $g_0^*$ to optimize (8)
 5: $f(x) = g_0^*$ for $t = 1, 2, \cdots, N_b$ do
 6:
          end
          update importance index $I_{j,k}$ for $(j, k) \in P_{t-1}$, using (1)
 7: select the subgrid based on $\sigma$ fraction of rows and columns with highest importance
    index and let $P_t$ be the set of selected pixels; form a new tensor $x_i^t$ for each sample $i$
 8: construct a new proper weak learner architecture
 9: compute $w(x_i, z_i)$ for all $i$, using (6) and (3)
10: train a deep CNN $g_t^*$ to optimize (2)
11: find the optimal coefficient $\alpha_t$, using (9) and (3)
12: $f(x) = f(x) + \nu \alpha_t g_t^*$
13: end for
```

Subgrid BoostCNN starts by initializing $f(x) = \mathbf{0} \in \mathbb{R}^M$. The algorithm first generates a full-size deep CNN as the basic weak learner, which uses the full image in steps 3-4. After the basic weak learner g_0^* is generated, in each iteration, subgrid BoostCNN first updates the importance index $I_{j,k}$ for each pixel (j, k), which has been used in the preceding iterate at step 6. In order to mimic the loss of the full-size image, although we only update the importance indexes for the pixels which have been used in the last iterate, we feed the full-size tensor to the deep CNN g to compute the importance index. The deep CNN g used in (1) to compute the importance value is constructed by copying the feature extractor from the preceding weak learner followed by the classifier in the basic weak learner g_0^*. Next, by deleting less important rows and columns based on $I_{j,k}$, which contain $1 - \sigma$ fraction of pixels, it finds the most important subgrid having σ fraction of pixels at position P_t based on the importance index $I_{j,k}$, and forms a new tensor x_i^t in step 7. Note that P_t is not necessary to be a subset of P_{t-1} and actually is rarely to be a subset of P_{t-1}. This only happens when the highest importance index at iterate t is also the highest score at iterate $t - 1$. Next, a new additive weak learner is initialized by borrowing the feature extractor from the preceding weak learner g_{t-1}^* followed by a randomly initialized FC layer with the proper size in step 8. Once the additive weak learner is initialized, subgrid BoostCNN computes the boosting weights, $w(x) \in \mathrm{R}^M$ according to (6) and (3), trains a network g_t^* to minimize the squared error between the network output and boosting weights using (2), and finds the boosting coefficient α_t by minimizing the boosting loss (9) in steps 9-11. Lastly, the algorithm adds the network to the ensemble according to $f(x) = f(x) + \nu \alpha_t g_t^*$ for $\nu \in [0, 1]$ in step 12.

3. Experimental Study

In this section, we apply the subgrid trick on both standard **BoostCNN** Moghimi et al. (2016) and an **ensemble of CNNs (e-CNN)**, where multiple CNNs are trained independently and their predictions are averaged without boosting weights or feature selection. We analyze the impact of *boosting*, *subgrid selection*, and *importance sampling*. All models are implemented in PyTorch Paszke et al. (2017) and trained on an NVIDIA Titan XP GPU. Subgrid BoostCNN uses cross-entropy loss and standard 3-channel image inputs handled by `Conv2d`. The subgrid strategy, based on Equation (1), drops approximately 10% of rows and columns, retaining about 81% of the pixels. We fix the shrinkage parameter to $\nu = 0.02$ and train each weak learner using the ADAM optimizer with a learning rate and weight decay of 0.0001.

We consider CIFAR-10 Krizhevsky et al. (2009), SVHN Netzer et al. (2011) and ImageNetSub Deng et al. (2009) datasets as shown in Table 1. For the last dataset, since the original ImageNet dataset is large and takes significant amount of time to train, we select a subset of samples from the original ImageNet dataset. More precisely, we randomly pick 100 labels and select the corresponding samples from ImageNet, which consists of $124,000$ images for training and $10,000$ images for testing. We denote it as ImageNetSub.

Dataset	Number of Training/Number of Testing	Number of Classes
CIFAR-10	50k/10k	10
SVHN	73k/26k	10
ImageNetSub	124k/10k	100

Table 1: Image datasets used in our experiments.

We evaluate three CNN architectures—ResNet-18, ResNet-50, and ResNet-101—across multiple datasets. For each dataset/architecture pair, we pretrain the CNN and use its weights to initialize the base weak learner. In Subgrid BoostCNN, we train 10 weak learners for 15 epochs each. For comparison, we consider (i) **e-CNN**, an ensemble of independently trained CNNs aggregated without boosting or feature selection, and (ii) **subgrid e-CNN**, which applies the subgrid trick without boosting weight updates (step 9 in Algorithm 1), resembling random forests. Both ensemble baselines are trained for 10 iterations. Finally, we train a single CNN for 150 epochs to approximate the same total compute as Subgrid BoostCNN. We apply ResNet-18 as our weak learner for all different ensemble methods. Figures 1, 3 and 5 compare the relative performances with respect to single ResNet-18 vs the running time. The solid lines in green and yellow show the relative performances of BoostCNN and subgrid BoostCNN, respectively, while the dotted lines in green and yellow represent the relative performances of e-CNN and subgrid e-CNN, respectively. As shown in these figures, taking the same amount of time, subgrid BoostCNN outperforms all of the remaining algorithms. We observe that subgrid BoostCNN outperforms BoostCNN, and subgrid e-CNN has the same behavior when compared with e-CNN. Moreover, Figures 2, 4 and 6 depict subgrid BoostCNN and subgrid e-CNN using three different seeds with respect to their averages. The solid and dotted lines in the same color represent the same seed used in corresponding subgrid BoostCNN and subgrid e-CNN. As the figures show, the solid lines are closer to each other than the dotted lines, which indicates that subgrid BoostCNN is more robust with respect to the variation of the seed when compared with subgrid e-CNN. Furthermore, the standard deviations of the accuracy generated by subgrid e-CNN and subgrid BoostCNN are shown in Table 2. The standard deviations of the accuracy generated by subgrid e-CNN are significant compared to those of subgrid BoostCNN, which in turn indicates that subgrid BoostCNN is less sensitive to the choice of the seed. Therefore, subgrid BoostCNN is more robust than subgrid e-CNN.

	subgrid BoostCNN	subgrid e-CNN
CIFAR-10	0.478	2.519
SVHN	0.385	0.891
ImageNetSub	2.489	7.915

Table 2: Standard deviation times 10^3 of the accuracy results by different seeds

Next, we evaluate relative performances of subgrid BoostCNN using ResNet-50 as the weak learner on CIFAR-10 and ImageNetSub datasets with respect to the single ResNet-50. We do not evaluate the relative performances on the SVHN dataset since the accuracy of the single ResNet-50 on the SVHN dataset is over 98%. From Figures 7 and 9, we also observe the benefits of the subgrid technique. Besides, Figures 8 and 10 confirm that subgrid BoostCNN is more stable than subgrid e-CNN since the solid series are closer to each other compared with the dotted series. Furthermore, we establish the relative performances of subgrid BoostCNN using ResNet-50 as the weak learner with respect to the single ResNet-101 in Figure 11. Although single ResNet-101 outperforms single ResNet-50, subgrid BoostCNN using ResNet-50 as the weak learner outperforms single ResNet-101 significantly in Figure 11, which indicates that subgrid BoostCNN with a simpler CNN is able to exhibit a better performance than a single deeper CNN. Lastly, we conduct experiments with ResNet-101 on the ImageNetSub dataset. From Figure 12, we not only discover the superior behaviors of BoostCNN, e-CNN, subgrid BoostCNN and subgrid e-CNN over ResNet-101 as we expect, but also observe the benefit of the subgrid technique.

4. Conclusion

We presented Subgrid BoostCNN, a boosting framework that integrates gradient-based subgrid selection to reduce computation while maintaining or improving classification performance. Our results show that Subgrid BoostCNN consistently achieves higher accuracy and lower variance than its non-subgrid counterparts. Specifically, with 10 ResNet-18-based weak learners, Subgrid BoostCNN outperforms both BoostCNN and e-CNN across all datasets for the same total training time. It improves accuracy by up to 12.10% over the base CNN and 4.19% over BoostCNN, while reducing sensitivity to random initialization. Standard deviation analysis further reveals Subgrid BoostCNN's robustness, especially when compared to subgrid e-CNN, which suffers from higher variance. Additionally, we find that Subgrid BoostCNN generalizes well across architectures. Using ResNet-50 as the base learner, it still outperforms deeper single CNNs like ResNet-101. This suggests that Subgrid BoostCNN can deliver better accuracy even with shallower models, which is particularly beneficial for large-scale or resource-constrained scenarios.

References

Lorenz Berger, Eoin Hyde, Matt Gibb, Nevil Pavithran, Garin Kelly, Faiz Mumtaz, and Sébastien Ourselin. Boosted training of convolutional neural networks for multi-class segmentation. *ArXiv*, abs/1806.05974, 2018.

Sourour Brahimi, Najib Ben Aoun, and Chokri Ben Amar. Boosted convolutional neural network for object recognition at large scale. *Neurocomputing*, 330:337–354, 2019.

Quang-Thanh Bui, Tien-Yin Chou, Thanh-Van Hoang, Yao-Min Fang, Ching-Yun Mu, Pi-Hui Huang, Vu-Dong Pham, Quoc-Huy Nguyen, Do Thi Ngoc Anh, Van-Manh Pham, and Michael E. Meadows. Gradient boosting machine and object-based cnn for land cover classification. *Remote Sensing*, 13(14), 2021.

Dominik Csiba and Peter Richtárik. Importance sampling for minibatches. *ArXiv*, abs/1602.02283, 2018.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009.

Seyedsaman Emami and Gonzalo Martínez-Muñoz. A gradient boosting approach for training convolutional and deep neural networks. *IEEE Open Journal of Signal Processing*, 4:313–321, 2023.

Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, and Jianfei Cai. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.

Shizhong Han, Zibo Meng, Ahmed-Shehab Khan, and Yan Tong. Incremental boosting convolutional neural network for facial action unit recognition. In *NIPS*, 2016.

Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class AdaBoost. *Statistics and Its Interface*, 2:349–360, 2009.

Angelos Katharopoulos and François Fleuret. Not all samples are created equal: Deep learning with importance sampling. *ArXiv*, abs/1803.00942, 2018.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Citeseer*, 2009.

Shin-Jye Lee, Tonglin Chen, Lun Yu, and Chin-Hui Lai. Image classification based on the boost convolutional neural network. *IEEE Access*, 6:12755–12768, 2018.

Mohammad Moghimi, Serge J. Belongie, Mohammad J. Saberian, Jian Yang, Nuno Vasconcelos, and Li-Jia Li. Boosted convolutional neural networks. In *BMVC*, 2016.

Indraneel Mukherjee and Robert E Schapire. A theory of multiclass boosting. *Journal of Machine Learning Research*, 14:437–497, 2013.

Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics*, 7, 2013.

Deanna Needell, Rachel Ward, and Nati Srebro. Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. *Mathematical Programming*, 155:549–573, 2014.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS*, 2011.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. *NIPS*, 2017.

Ross J. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 2004.

Mohammad J Saberian and Nuno Vasconcelos. Multiclass Boosting: Theory and algorithms. In *NIPS*, 2011.

Peilin Zhao and Tong Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In *ICML*, 2015.

Appendix A. Related Work

There are many extensions of Gradient Boosting Machine (GBM) Natekin and Knoll (2013), however, a full retrospection of this immense literature exceeds the scope of this work. In this section, we mainly state several kinds of variations of GBM which are most related to our new algorithms, together with the two add-ons to our optimization algorithms, i.e. subgrid and importance sampling.

Deep CNNs, which have recently produced outstanding performance in learning image representations, are capable of learning complex features that are highly invariant and discriminant Gu et al. (2018). The success of deep CNNs in recognizing objects has encouraged recent works to combine boosting together with deep CNNs. Brahimi & Aoun Brahimi et al. (2019) propose a new Boosted Convolutional Neural Network architecture, which uses a very deep convolutional neural network reinforced by adding Boosted Blocks which consist of a succession of convolutional layers boosted by using a Multi-Bias Nonlinear Activation function. Nevertheless, the architecture of the proposed Boosted convolutional neural network is fixed; it can not dynamically change the number of Boosted Blocks for a given dataset. Another attempt at combining deep CNNs and boosting is boosted sampling Berger et al. (2018), which uses posterior error maps, generated throughout training, to focus sampling on different regions, resulting in a more informative loss. However, boosted sampling applies boosting on selected samples and treats deep CNN as a black box to make a prediction. To enrich the usage of the information generated by deep CNNs, Lee & Chen Lee et al. (2018) propose a new BoostCNN structure which employs a trained deep convolutional neural network model to extract the features of the images and then use the AdaBoost algorithm to assemble the Softmax classifiers. However, it remains unclear how to combine different sets of the features extracted and the computational cost is high when training several deep CNNs at the same time. To tackle this problem, Han & Meng Han et al. (2016) propose Incremental Boosting CNN (IB-CNN) to integrate boosting into the CNN via an incremental boosting layer that selects discriminative neurons from a lower layer and is incrementally updated on successive mini-batches. Different from IB-CNN which only involves one deep CNN, BoostCNN Moghimi et al. (2016) incorporates boosting weights into the neural network architecture based on least squares objective functions, which leads to the aggregation of several CNNs. However, the computational and memory demand of BoostCNN is high when the weak learner is not simple. To enrich the usage of the information generated by deep CNNs, Bui et al. Bui et al. (2021) explored the integration of gradient boosting algorithms (GBMs) with Object-Based CNN. Their approach replaces the fully connected layers of the CNN model with GBMs (XGBoost, LightGBM, and CatBoost), effectively combining the feature extraction power of CNNs with the robust classification capabilities of GBMs. ConvXGB-2021 is a deep learning model that integrates CNNs with eXtreme Gradient Boosting (XGBoost). In this framework, CNNs are employed for feature extraction, and the extracted features are then fed into an XGBoost classifier for final predictions.

Gradient Boosted Convolutional Neural Network (GB-CNN) Emami and Martínez-Muñoz (2023) iteratively adds a dense layer to an exact copy of the previous model, while keeping the weights of earlier dense layers frozen. This method helps prevent overfitting, fine-tune previously trained convolutional layers, and utilize learned information to improve classification performance. All of the above train the weak learners on all features. Given the fact that importance sampling improves the performance by prioritizing training samples, importance sampling has been well studied, both theoretically and empirically, in standard stochastic gradient descent settings Needell et al. (2014) Zhao and Zhang (2015), in deep learning settings Katharopoulos and Fleuret (2018), and in minibatches Csiba and Richtárik (2018). As stated in these papers, importance sampling theoretically improves the convergence rate and is experimentally effective in reducing the training time and training loss. However, no generalization work has been done in a boosting setting.

Appendix B. Background (BoostCNN):

We start with a brief overview of multiclass boosting. Given a sample $x_i \in \mathcal{X}$ and its class label $z_i \in \{1, 2, \cdots, M\}$, multiclass boosting is a method that combines several multiclass predictors $g_t : \mathcal{X} \to \mathbb{R}^d$ to form a strong committee $f(x)$ of classifiers, i.e. $f(x) = \sum_{t=1}^{N} \alpha_t g_t(x)$ where g_t and α_t are the weak learner and coefficient selected at the t^{th} boosting iteration. There are various approaches for multiclass boosting such as Hastie et al. (2009), Mukherjee and Schapire (2013), Saberian and Vasconcelos (2011); we use the GD-MCBoost method of Saberian and Vasconcelos (2011), Moghimi et al. (2016) herein. For simplicity, in the rest of the paper, we assume that $d = M$.

Standard BoostCNN Moghimi et al. (2016) trains a boosted predictor $f(x)$ by minimizing the risk of classification

$$\mathcal{R}[f] = \mathrm{E}_{X,Z}\left[L(z, f(x))\right] \approx \frac{1}{|\mathcal{D}|} \sum_{(x_i, z_i) \in \mathcal{D}} L(z_i, f(x_i)), \tag{4}$$

where $\mathcal{D}$ is the set of training samples and

$$L(z, f(x)) = \sum_{j=1, j\neq z}^{M} e^{-\frac{1}{2}[\langle y_z, f(x)\rangle - \langle y_j, f(x)\rangle]},$$

given $y_k = \mathbb{1}_k \in \mathbb{R}^M$, i.e. the k^{th} unit vector. The minimization is via gradient descent in a functional space. Standard BoostCNN starts with $f(x) = \mathbf{0} \in \mathbb{R}^d$ for every x and iteratively computes the directional derivative of risk (4), for updating $f(x)$ along the direction of $g(x)$

$$\begin{aligned}\delta\mathcal{R}[f;g] &= \left.\frac{\partial\mathcal{R}[f+\epsilon g]}{\partial\epsilon}\right|_{\epsilon=0}\\ &= -\frac{1}{2\,|\mathcal{D}|}\sum_{(x_i,z_i)\in\mathcal{D}}\sum_{j=1}^{M} g_j(x_i)w_j(x_i,z_i)\\ &= -\frac{1}{2\,|\mathcal{D}|}\sum_{(x_i,z_i)\in\mathcal{D}} g(x_i)^T w(x_i,z_i),\end{aligned} \tag{5}$$

where

$$w_k(x,z) = \begin{cases} -e^{-\frac{1}{2}[f_z(x)-f_k(x)]}, & k \neq z\\ \sum_{j=1,j\neq k}^{M} e^{-\frac{1}{2}[f_z(x)-f_j(x)]}, & k = z,\end{cases} \tag{6}$$

and $g_j(x_i)$ computes the directional derivative along $\mathbb{1}_j$. Then, standard BoostCNN selects a weak learner g^* that minimizes (5), which essentially measures the similarity between the boosting weights $w(x_i,z_i)$ and the function values $g(x_i)$. Therefore, the optimal network output $g^*(x_i)$ has to be proportional to the boosting weights, i.e.

$$g^*(x_i) = \beta w(x_i,z_i), \tag{7}$$

for some constant $\beta > 0$. Note that the exact value of β is irrelevant since $g^*(x_i)$ is scaled when computing α^*. Consequently, without loss of generality, we assume $\beta = 1$ and convert the problem to finding a network $g(x) \in \mathbb{R}^M$ that minimizes the square error loss

$$\mathcal{L}(w,g) = \sum_{(x_i,z_i)\in\mathcal{D}} \|g(x_i) - w(x_i,z_i)\|^2 . \tag{8}$$

After the weak learner is trained, BoostCNN applies a line search to compute the optimal step size along g^*,

$$\alpha^* = \underset{\alpha\in\mathbb{R}}{\arg\min}\,\mathcal{R}[f+\alpha g^*]. \tag{9}$$

Finally, the boosted predictor $f(x)$ is updated as $f = f + \alpha^* g^*$.

Appendix C. Experimental Results

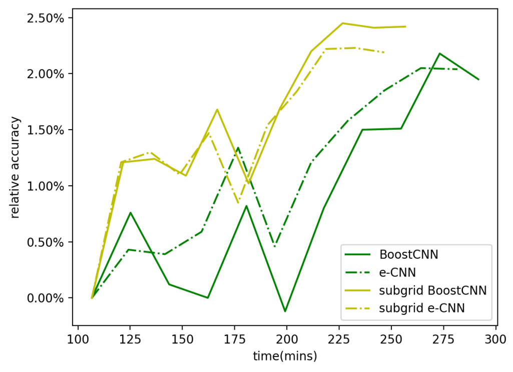

Figure 1: ResNet-18 on CIFAR-10

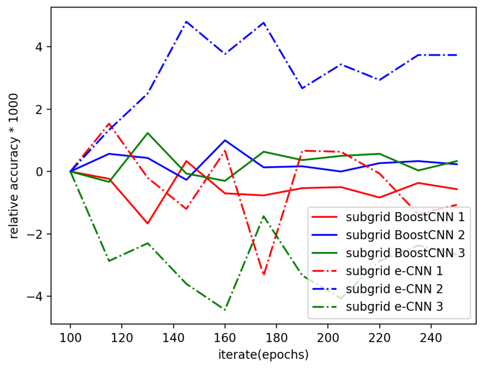

Figure 2: Different Seeds

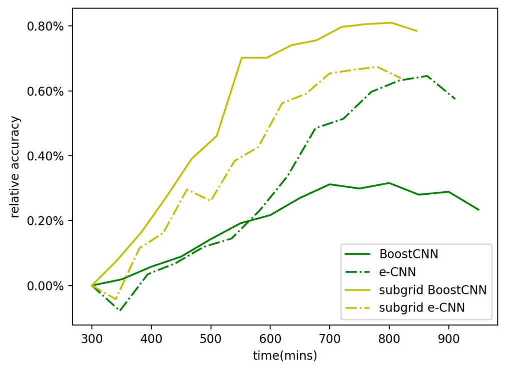

Figure 3: ResNet-18 on SVHN

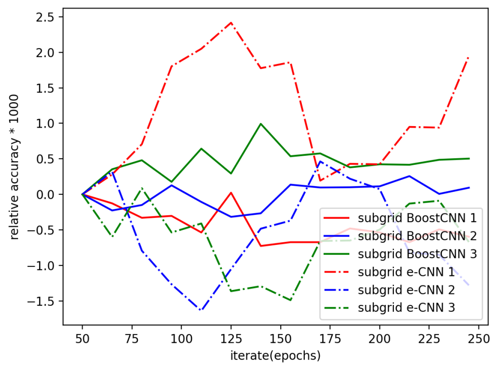

Figure 4: Different Seeds

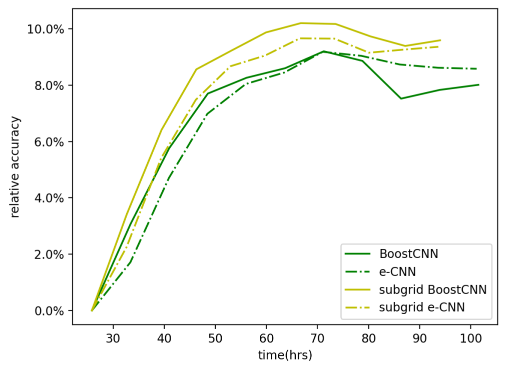

Figure 5: ResNet-18 on ImageNetSub

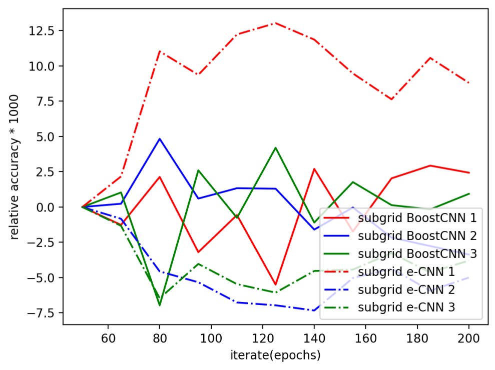

Figure 6: Different Seeds

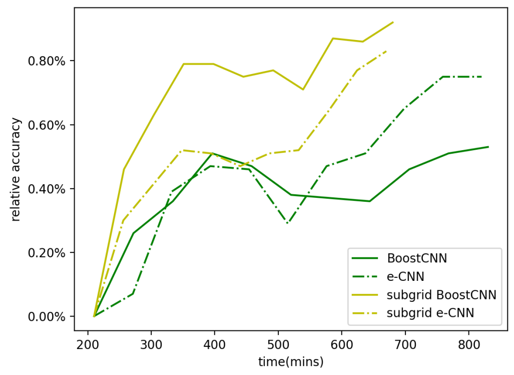

Figure 7: ResNet-50 on CIRFAR-10

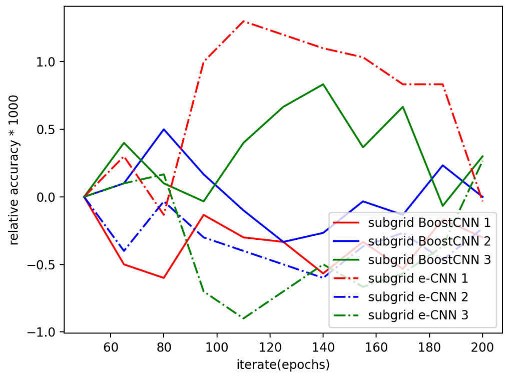

Figure 8: Different Seeds

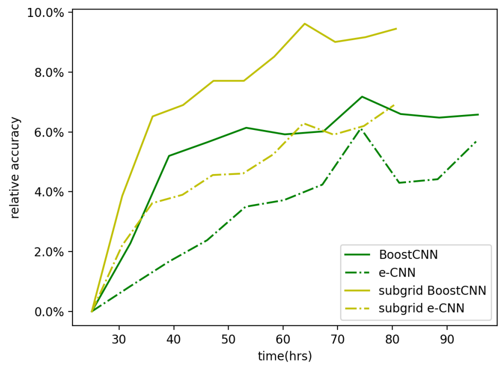

Figure 9: ResNet-50 on ImageNetSub

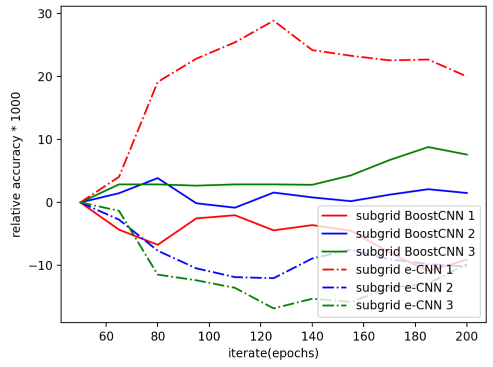

Figure 10: Different Seeds

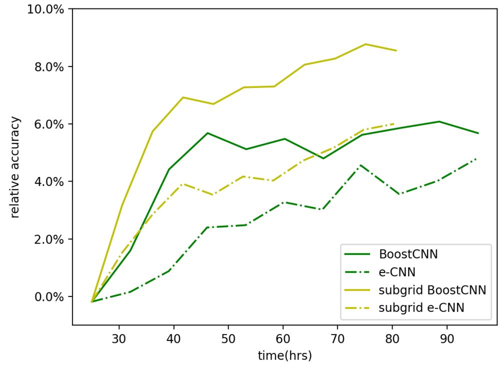

Figure 11: ResNet-50 on ImageNetSub compared to ResNet-101

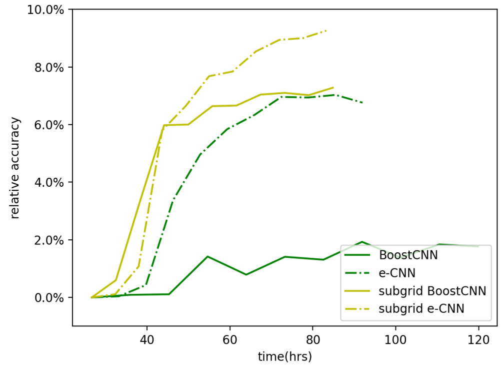

Figure 12: ResNet-101 on ImageNetSub