Dynamic Network Pruning with Interpretable Layerwise Channel Selection

Yulong Wang,[1] **Xiaolu Zhang,**[2] **Xiaolin Hu,**[1] **Bo Zhang,**[1] **Hang Su**[1]*
[1]Tsinghua University [2]Ant Financial
wang-yl15@mails.tsinghua.edu.cn, yueyin.zxl@antfin.com
{xlhu, dcszb, suhangss}@mail.tsinghua.edu.cn

Abstract

Dynamic network pruning achieves runtime acceleration by dynamically determining the inference paths based on different inputs. However, previous methods directly generate continuous decision values for each weight channel, which cannot reflect a clear and interpretable pruning process. In this paper, we propose to explicitly model the discrete weight channel selections, which encourages more diverse weights utilization, and achieves more sparse runtime inference paths. Meanwhile, with the help of interpretable layerwise channel selections in the dynamic network, we can visualize the network decision paths explicitly for model interpretability. We observe that there are clear differences in the layerwise decisions between normal and adversarial examples. Therefore, we propose a novel adversarial example detection algorithm by discriminating the runtime decision features. Experiments show that our dynamic network achieves higher prediction accuracy under the similar computing budgets on CIFAR10 and ImageNet datasets compared to traditional static pruning methods and other dynamic pruning approaches. The proposed adversarial detection algorithm can significantly improve the state-of-the-art detection rate across multiple attacks, which provides an opportunity to build an interpretable and robust model.

Introduction

In recent years, as deep neural networks have achieved significant performance in many fields (He et al. 2016; Silver et al. 2017), deep models have been deployed in many practical applications. However, due to the huge consumption of computation resources by the original full model, many model pruning and acceleration methods (Han, Mao, and Dally 2016; He, Zhang, and Sun 2017; Liu et al. 2017) have been proposed to meet the real-time computing requirements while ensuring prediction performance. Most of the pruning methods utilize existing pre-trained models as the basis and delete the redundant parameters and structures to obtain the final static compressed models. Pruned model weights are often fine-tuned based on the full model weights to improve model prediction performance further.

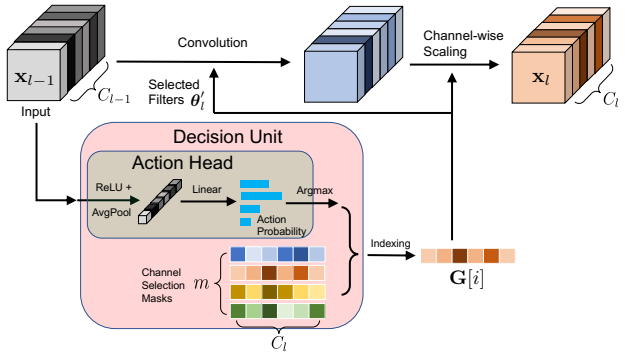

Figure 1: Overview of the proposed dynamic network pruning scheme. We associate each convolution layer with a decision unit, which is composed of action head and channel selection masks. For the l-th convolution layer in the network, the input $\mathbf{x}_{l-1}$ first passes through action head $\mathcal{A}$ to generate the action probability corresponding to a group of channel selection masks $\mathbf{G}$. Then the chosen channel mask $\mathbf{G}[i]$ is used to select convolution weights θ', which can achieve runtime compression and acceleration. The channel mask values are also multiplied to convolution output to perform channel scaling. We use continuous relaxation to resolve non-differentiability in action head, and all the parameters of decision units are trained in an end-to-end manner, which allows channel masks to learn the filters selection automatically.

Different from the above static pruning techniques, dynamic network pruning (Lin et al. 2017; Gao et al. 2018) determines the inference paths to achieve runtime acceleration according to different inputs. Compared to the static pruning, dynamic pruning methods preserve the full model but add runtime decision units inside each computational layer, which can complete the feedforward operations with a portion of the network weights by generating different weight masks. However, previous methods (Gao et al. 2018; Luo and Wu 2018) generate continuous gate predictions by directly outputting each channel's decision gate value, which cannot reflect a transparent and interpretable pruning pro-

*Corresponding Author

cess. Moreover, the resulting pruning masks tend to be homogeneous without input-dependent diversity, which nearly degenerates to static pruning in effect. This not only limits the degree of compression ratio but also affects the deployment of dynamic pruning techniques in the practical application scenarios.

In this paper, we propose to explicitly model the discrete weight channel decisions during the dynamic pruning process. The main idea is to associate a decision unit with each computation layer (*i.e.*, convolution layer in the CNN), which outputs a probability distribution corresponding to a finite number of different channel selection masks. During the model inference in the test stage, the channel mask corresponding to the maximum probability is selected as the final channel selection result. We utilize continuous relaxation technique (Maddison, Mnih, and Teh 2016; Jang, Gu, and Poole 2016) to resolve the non-differentiability issue during training the action head. The channel selections for each action are automatically learned under the sparsity regularization. The decision unit and channel mask values are jointly learned in an end-to-end manner.

Since our method can obtain discrete decision actions on each layer for input examples, this provides an interpretable representation to understand the model's functional behavior and prediction results. We observe that the inputs' layerwise decisions are highly correlated with the semantic meaning of their category. Meanwhile, we observe that the layerwise decision paths of adversarial examples (Szegedy et al. 2013) are different from those of normal examples, which reflects the different network response patterns between normal and adversarial examples. This phenomenon inspires us to develop a novel adversarial example detection algorithm that uses the layerwise decision features to discriminate whether a testing sample is normal or adversarial.

We conduct extensive experiments on CIFAR10, CIFAR100, SVHN and ImageNet datasets. Our dynamically pruned model achieves higher prediction accuracy than state-of-the-art pruning methods under the same pruning ratio. Our adversarial detection method significantly improves the performance of adversarial example detection across different attacks. These results demonstrate that our method provides an opportunity to build an efficient, interpretable and robust model.

Related Work

The model pruning techniques aim to remove the internal redundancy parameters of the model, and thus expect to achieve the acceleration of the deep neural network inference. Early works (LeCun, Denker, and Solla 1990; Han, Mao, and Dally 2016) focus on non-structured sparsity by removing individual weight values. Although runtime acceleration can be achieved through a custom inference engine, it is not friendly to general-purpose GPUs. In recent years, model pruning techniques have mainly focused on the development of structured pruning (Li et al. 2016; He, Zhang, and Sun 2017; Liu et al. 2017; He et al. 2018b), which prunes the weight channels of each layer. Pruned models often require further fine-tuning to achieve higher prediction performance. However, some works (Liu et al. 2019) have pointed out that the compressed model structure can be trained from scratch to achieve good performance without relying on pre-trained weights.

In addition to the above static pruning techniques, dynamic pruning techniques have also been proposed to achieve more flexible computation process. The Runtime Network Pruning (RNP) (Lin et al. 2017) selects the weight channel group on-the-fly, and each layer relies on a global decision agent to output discrete actions. Feature Boosting & Suppression (FBS) (Gao et al. 2018) uses the local decision unit to output the continuous channel importance values directly. It also uses a pre-defined compression ratio to sort the mask value to determine the channel selection. The most similar work to our approach is RNP. However, RNP predefines four channel groups across all the layers, while our method automatically learns the channel selection during training. This provides more diverse runtime pruning behaviors.

Another application of the method described herein is adversarial example detection. The adversarial examples (Szegedy et al. 2013) can change the model prediction through small perturbations on input, which poses a significant threat to the security and credibility of the deep models. Adversarial detection methods are thus proposed to build robust machine learning system. Typical methods include detection through the auxiliary network (Metzen et al. 2017), or utilizing adversarial training technique (Tramèr et al. 2017) to achieve defense. The proposed approach follows the detection methods based on the confidence score (Feinman et al. 2017; Ma et al. 2018; Lee et al. 2018). These methods calculate confidence value to judge whether the testing sample is on the data manifold formed by the normal samples, thus used for detecting abnormal examples.

Methodology

The overview of our dynamic network pruning scheme is illustrated in Figure 1. When the network performs feedforward operations for a single input, it will first output the weight channel masks through the decision unit, and then determine the weights used to perform actual calculation. The decision unit outputs a probability distribution corresponding to a finite number of different channel selection masks. During inference, the channel mask corresponding to the maximum probability is selected to construct the actual weights used for computation. Moreover, the channel selection values are also multiplied to convolution output channel-wisely to adjust the final output. Since the compressed weights are determined *before* the convolution operation, the actual computation cost (i.e., multiply-accumulate operations (MACs)) can be effectively reduced.

In the following sections, we will elaborate on the problem formulation for how to implement the proposed dynamic pruning scheme, and then present the details for each component.

Problem Formulation

The overall training objective for the proposed dynamic pruning scheme is

$$\min_{\{\mathbf{\Theta},\mathbf{\Phi}\}} \mathcal{L} = \sum_k \mathcal{L}_{ent}(f_{\mathbf{\Theta}}(\mathbf{x}_k), y_k) + \gamma \cdot \Omega\left(\{\mathbf{G}_l\}_L\right), \quad (1)$$

where $\mathbf{\Theta}$ denotes convolution layer weights, $\mathbf{\Phi}$ denotes all the parameters of decision heads $\theta_{\mathcal{A}}$ and channel selection masks $\{\mathbf{G}_l\}_L$ across different layers, $\mathcal{L}_{ent}$ is the cross-entropy loss between model prediction $f(\mathbf{x}_k)$ and label y_k, Ω is the regularization term to encourage channel selection masks to be sparse with a balance factor γ. The optimization in Equation (1) can be divided into two sub-problems for $\mathbf{\Theta}$ and $\mathbf{\Phi}$, and an alternate training strategy is adopted to solve the optimization. In the experiments, we use Adam and proximal gradient descent optimizers for learning $\mathbf{\Phi}$ and $\mathbf{\Theta}$ respectively.

Decision Unit Formulation

In a convolutional neural networks with L layers, the l-th convolutional layer computes the output features $\mathbf{x}_l \in \mathbb{R}^{N\times C_o\times H_o\times W_o}$ with input $\mathbf{x}_{l-1} \in \mathbb{R}^{N\times C_i\times H_i\times W_i}$ and weights $\theta_l \in \mathbb{R}^{C_o\times C_i\times k\times k}$ by

$$\mathbf{x}_l = \mathrm{conv}(\mathbf{x}_{l-1}, \theta_l), \quad (2)$$

where (N, C, H, W) correspond to (batch, channel, height, width) dimensions, subscripts i, o denotes variables for input and output, k is kernel size and conv is the convolution operation. For simplicity, we omit layer subscript in the following derivation.

To perform dynamic weight channel selection, a decision unit $\pi(\mathbf{x}, \phi)$ is associated with each convolution layer, where ϕ is its parameters. In RNP (Lin et al. 2017), the decision unit is modeled as a global recurrent layer, which receives input features to generate discrete actions corresponding to four preset channel selection groups. In FBS (Gao et al. 2018), the decision unit is modeled as local linear layers, which receives input features to generate the masks for each individual channel.

In our method, we model $\pi(\cdot)$ as follows

$$\pi(\mathbf{x}, \phi) = \mathbf{G}[i], \quad \text{where} \quad i = \arg\max \mathcal{A}(\mathbf{x}), \quad (3)$$

where $\mathbf{G} = \{g_i \in \mathbb{R}^{C_l} | i = 1 \cdots m\}$ is a set of channel mask vectors, each of which contains C_l values equal to output channel number; $\mathcal{A}(\cdot)$ is the discrete action head, which can output the probability $\mathbf{p} \in \mathcal{R}^m$ for m channel masks. Then the final output for l-th convolution layer is

$$\mathbf{x}_l = \mathrm{conv}(\mathbf{x}_{l-1}, \theta_l) * \pi(\mathbf{x}_{l-1}, \phi), \quad (4)$$

where $*$ is channel-wise multiplication for scaling output. To reduce the actual computation cost, one can first select weights with non-zero channel selection values $\theta_l^{'} = \{\theta_l[j] | \mathbf{G}[i][j] \neq 0\}$, and then perform the convolution. $\mathcal{A}(\cdot)$ is implemented as

$$\mathcal{A}(\mathbf{x}) = \texttt{Linear}(\texttt{GlobalAvgPool}(\texttt{ReLU}(\mathbf{x}))). \quad (5)$$

Since the decision unit is designed to be lightweight, the overall computation cost is still highly reduced compared to the original full model.

Differentiable Training of Action Head

Since the output of action head $\mathcal{A}(\cdot)$ is discrete, it is usually trained with policy gradient (Williams 1992) technique developed in reinforcement learning. However, policy gradient suffers from high variance and training instability (Sutton, Barto, and others 1998). Here we utilize continuous relaxation (Maddison, Mnih, and Teh 2016; Jang, Gu, and Poole 2016) reparameterization technique to directly train the action head in a differentiable manner.

Specifically, given the action head output probability $\mathbf{p} = [p_1, \cdots, p_m]$ for m actions, the chosen index I for channel selection mask is modeled as a categorical random variable with $\mathbb{P}(I = i) \propto p_i$. The sampling process can be reparameterized as

$$I = \arg\max_i \log p_i + G_i, \quad (6)$$

where G_i is a sequence of *i.i.d.* Gumbel random variables (Gumbel 1954), sampled from the Gumbel distribution $G = -\log(-\log(X))$ with $X \sim U[0, 1]$. To deal with the non-differentiability to the underlying probability $\mathbf{p}$, the one-hot representation of I is replaced with a softmax form (Maddison, Mnih, and Teh 2016; Jang, Gu, and Poole 2016), which is expressed as

$$I_i = \frac{\exp((\log p_i + G_i)/\tau)}{\sum_{j=1}^m \exp((\log p_j + G_j)/\tau)}, \quad \forall\, i = 1, \cdots, m, \quad (7)$$

where τ is a temperature to control the probability concentration degree. During forward, the chosen channel mask is obtained by matrix multiplication without resorting to indexing table.

Learning Channel Selection

One merit of our method is that the channel selection masks $\mathbf{G}$ are automatically learned in an end-to-end manner without pre-defined configurations adopted in RNP (Lin et al. 2017). During training, the masks $\mathbf{G}$ are influenced by both classification loss $\mathcal{L}_{ent}$ and regularization term Ω. As for the concrete form of Ω, an ℓ_1-norm is utilized to encourage sparse channel selection masks. However, only ℓ_1-norm cannot control the sparsity ratio. Similar to (Luo and Wu 2018), we use the element-wise mean of the concatenated channel masks to approximate the overall compression ratio, and use a square norm to constrain the compression ratio to a target vale r. Specifically, given a target sparsity ratio r, the regularization term Ω is

$$\Omega\left(\{\mathbf{G}_l\}_L\right) = \left(\frac{||\mathrm{concat}(\mathbf{G}_1, \cdots, \mathbf{G}_L)||_1}{\sum_l C_l} - r\right)^2, \quad (8)$$

where concat is concatenation operation, C_l is the l-th channel number equal to the dimension of $\mathbf{G}_l$. Unlike RNP and FBS, we only define a global compression ratio r and the specific layer-wise channel selections are automatically learned during training. In the following experiment section, we will show that this procedure can learn diverse weight channel selections.

Implementation Details

Increasing Action Diversity During the initial exploration, we found that the action head in the decision unit always generated the same channel masks for different inputs, because some actions in the action head dominated the action predictions throughout the time. To alleviate this effect, we propose two simple techniques to increase action diversity, which are weight normalization and annealing temperature. After the weights in the action head $\theta_{\mathcal{A}}$ are updated in each iteration, we further normalize each weight vector $\mathbf{v}$ to be $\mathbf{v}' = \mathbf{v}/||\mathbf{v}||_2$, which makes the weights focus on learning directions. Moreover, we linearly decrease the temperature variable τ in Equation (7) from 5.0 to 0.5 along training process, expecting that the action head can sample uniform actions in the initial training phase and converge to a stable decision process in the end.

Decision Unit Location The decision unit can be associated with each computational layers in the network. For single path network like VGGNet, we associate the decision unit behind each convolution layer's output except the first convolution layer. For networks with skip connections like ResNet, we associate the decision units behind the convolution layers on the residual branch, except the last convolution layer in the residual block, since it can ensure that the output dimension is consistent with input dimension at the beginning of the block.

Experiments

In this section, we conduct experiments on CIFAR10 and ImageNet datasets. We perform dynamic pruning on VGGNets and ResNets. For CIFAR10 models, we train the full models for 160 epochs using a batch-size of 128 with SGD optimizer. The initial learning rate 0.1 is divided at 50% and 75% of the total number of training epochs. We use a momentum of 0.9 with weight decay of 10^{-4}. For ImageNet models, we use the pruned models VGG_5x and ResNet50_2x from Channel Pruning (He, Zhang, and Sun 2017) as the pre-trained weights to further accelerate optimization speed.

Settings

For the CIFAR10 experiments, we choose $m = 5$ actions for each decision unit after the trade-off between performance and computation cost. The target sparsity ratio r is 0.1 for VGG16-BN and 0.4 for ResNet56. Balance factor $\gamma = 1.0$, the learning rate is 0.01, training batch size is 128, and the total training epoch is 100. We further fine-tune the backbone network weights while freezing decision units to improve performance. The fine-tuning scheme is the same as that of pre-training, except that the starting learning rate is reduced to 0.001.

For the ImageNet experiments, we choose $m = 40$ actions for each decision unit. The target sparsity ratio r is 0.6 for VGG16 and ResNet50. Balance factor $\gamma = 1.0$, learning rate is 0.01, training batch size is 64. Since the starting model is compressed, we only need to train for 30 epochs. The fine-tuning scheme is same with the above settings.

Pruning Results

Table 1: Comparison of accuracy for different pruning methods with VGG16-BN and ResNet56 models on CIFAR10 dataset. "Pruning Ratio" column stands for the MACs reduction ratio, and higher is better.

	Method	MACs	Pruning Ratio	Acc
VGG16-BN	Baseline	313.2M	–	93.50%
	ThiNet	156.5M	50%	93.36%
	L1-norm	206.6M	34%	93.00%
	CP	156.5M	50%	93.18%
	NS	153.4M	51%	93.31%
	RNP	156.5M	50%	92.65%
	FBS	156.5M	50%	93.03%
	Ours	155.2M	50.4±0.04%	**93.45%**
ResNet56	Baseline	125.8M	–	92.80%
	ThiNet	62.9M	50%	91.98%
	CP	62.9M	50%	91.80%
	SFP	65.4M	48%	92.56%
	AMC	62.9M	50%	90.20%
	Ours	59.6M	**52.6±0.13%**	**92.57%**

Table 2: Comparison of top-5 accuracy for different pruning methods with VGG16 and ResNet50 models on ImageNet dataset. "Pruning Ratio" column stands for the MACs reduction ratio, and higher is better. "uniform" stands for uniformly reducing the channels of each layer with a fixed ratio.

	Method	MAC	Pruning Ratio	Top-5 Acc
VGG16	baseline	15.5G	–	89.90%
	L1-norm	3.8G	75%	86.54%
	CP	3.1G	80%	88.10%
	NS	3.1G	75%	84.72%
	AMC	3.1G	80%	88.50%
	RNP	3.1G	80%	86.32%
	Ours	2.9G	**81.2±0.01%**	**88.57%**
ResNet50	baseline	4.1G	–	92.93%
	uniform	2.0G	50%	74.39%
	CP	2.0G	50%	90.80%
	ThiNet	2.6G	37%	91.84%
	SFP	2.4G	41.80%	92.06%
	Ours	1.9G	**51.3±0.64%**	**92.08%**

We compare our method with both static and dynamic pruning techniques. For static pruning, we compare with ThiNet (Luo, Wu, and Lin 2017), ℓ_1-norm based filter pruning (Li et al. 2016), Channel Pruning (CP) (He, Zhang, and Sun 2017), Network Slimming (NS) (Liu et al. 2017), Soft Filter Pruning (SoftFilter) (He et al. 2018a), and Automatic Model Compression (AMC) (He et al. 2018b). For dynamic pruning, we compare with Runtime Neural Pruning (RNP) (Lin et al. 2017) and Feature Boosting & Suppression (FBS) (Gao et al. 2018). We report the average accuracy over

Table 3: Test accuracy and MACs reduction ratios for different numbers of actions used. For MACs reduction ratios, lower is better.

Action Number m		1	2	3	4	5	6	7	8	9	10
VGG16-BN	Acc (%)	93.38	93.09	93.11	93.09	93.45	93.20	93.05	93.29	93.15	93.13
	MACs Ratio (%)↓	51.90	48.69	48.49	50.19	49.58	50.26	50.92	49.59	49.66	47.94
ResNet56	Acc (%)	92.47	92.32	92.32	92.42	92.57	92.32	92.29	92.09	92.26	92.00
	MACs Ratio (%)↓	62.68	59.46	55.19	52.87	47.38	48.91	51.04	47.64	45.69	49.04

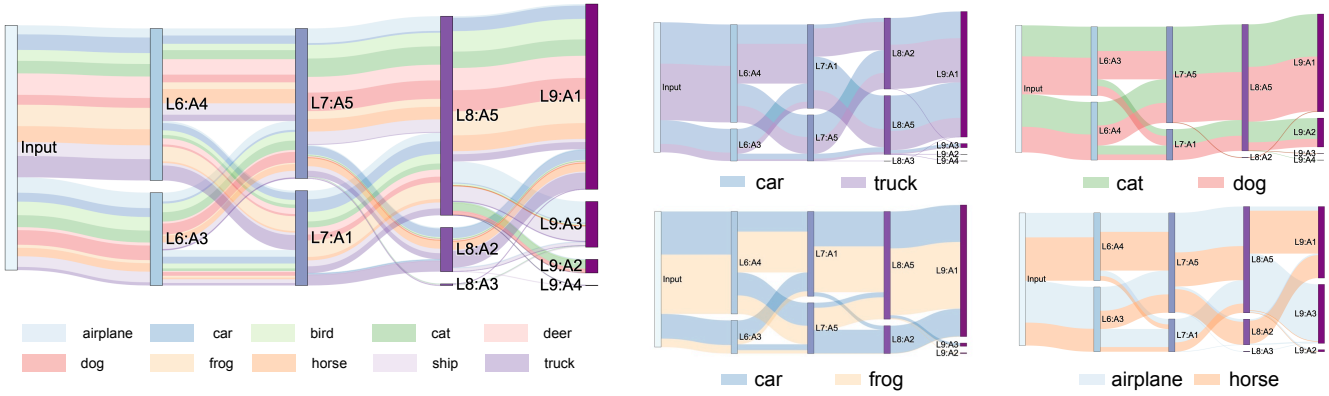

Figure 2: Runtime analysis of the decisions in dynamic pruning network. (Left) We use Sankey diagram to visualize the decision paths for CIFAR10 test samples between the sixth to ninth layers in VGG16-BN. Long strip node represents an action. Different colored strip links between action nodes represent the decision paths taken by the different categories of samples. (Right) We further use Sankey diagrams to visualize the decision paths for these categories pairs: (car, truck), (cat, dog), (car, frog) and (airplane, horse).

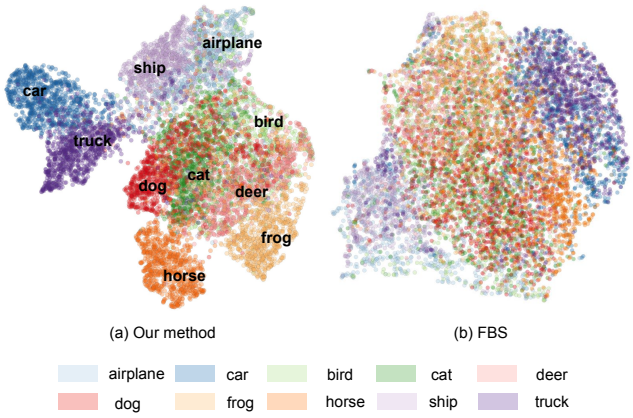

Figure 3: The decision feature embedding visualization.

five repeated experiments and MACs including the computation of decision units.

Table 1 summarizes the pruning results on CIFAR10 dataset. Our dynamic pruning models achieve the highest accuracy under the similar pruning ratios. Specifically, our dynamic model for ResNet56 reduces more computational cost than other pruning methods with the best performance. Moreover, our method reaches the highest prediction accuracy among all the dynamic pruning techniques.

Table 2 summarizes the pruning results on ImageNet dataset. Our method achieves the best performance with the least runtime computation cost across different models. Specifically, our model outperforms other pruning methods with RL-based complicated strategies, such as AMC and RNP. These results demonstrate that our method is scalable and effective even on large-scale dataset.

Effects of Action Number

In this section, we conduct ablation study to explore the effects of different numbers of actions used in the decision unit. Table 3 summarizes the results on CIFAR10 dataset. All models are trained with the identical settings mentioned above, except with different action numbers. From the table, we observe that for VGG16-BN, different action numbers result in similar MACs reduction ratios and model accuracy. For ResNet56, with more actions in the action head, the dynamic network achieves more MACs reduction during inference, but at the cost of prediction performance drops. This implies that we should trade off the prediction performance and computation cost when designing the action head in the application scenarios.

When we reduce the action number m to 1, it means only one set of channel selection mask is learned and the overall method degenerates to static network pruning. For VGG16-BN model, it achieves a 93.38% accuracy with a 51.9% MACs ratio. For ResNet56 model, it achieves a 92.47% ac-

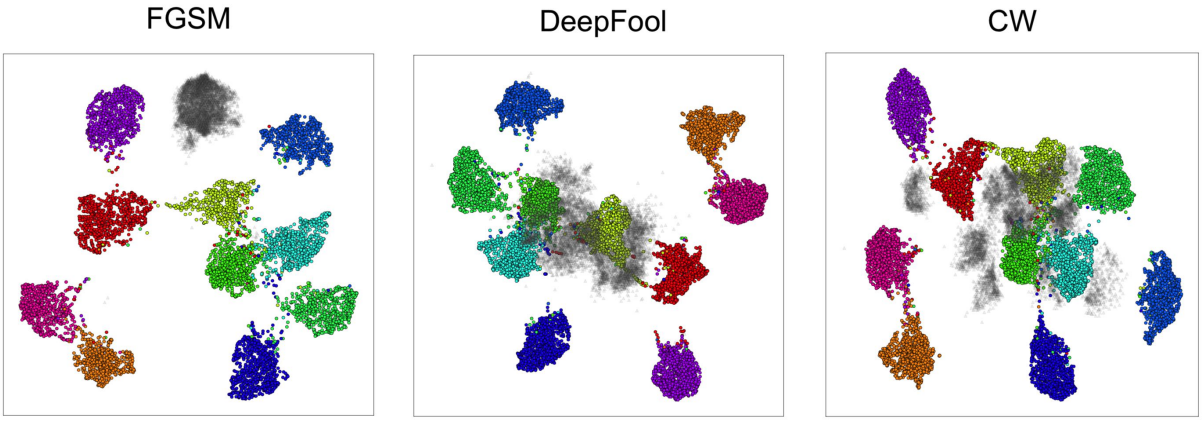

Figure 4: The decision feature embedding visualization for normal and adversarial examples on the CIFAR10 test dataset. In each figure, colored dots denote the embedding of normal examples' decision features, whose colors represent different category labels. The gray dots represent all the adversarial examples.

curacy with a 62.68% MACs ratio. These results demonstrate that conventional static pruning cannot achieve high computation reduction with high prediction performance. Therefore, we argue that our novel design for the dynamic pruning is necessary and effective.

Interpreting Model Runtime Decisions

In this section, we explore the possibility of the proposed dynamic model for model interpretability. We can visualize the inference paths (*e.g.,* the runtime layer-wise channel selections) utilized by different inputs to analyze the model's functional behavior when making predictions. To this end, we use the VGG16-BN model for investigation and collect the action head output on 11 convolutional layers[1] of the 10,000 samples in the CIFAR10 test dataset.

We use the Sankey diagram to visualize the decision paths for different samples. Figure 2 shows the visualization results. Each long strip node represents an action whose numbering rule is $L[i] : A[j]$, where i represents the number of layers and j represents the action index. Since some actions have never been adopted by any of the samples, they are not shown in the figure. Different colored strip links between action nodes represent the decision paths taken by the different categories of samples, the width of which are proportional to the ratio of the samples which take the action. It can be seen that similar semantic categories such as cats and dogs, cars and trucks have similar decision paths, that is, the distributions for different actions are similar. For classes with large semantic differences, such as airplanes and horses, there are significant differences in the decision paths. The detailed Sankey diagrams for two categories comparisons are depicted in the right figures in Figure 2. With the decision paths visualization, we can explicitly interpret the functional behavior of the deep models when making predictions.

[1]VGG16-BN has 12 convolution layers. We exclude the first layer from making decisions.

Comparison with Other Dynamic Pruning Method

In this section, we want to compare the inference behaviors of our method with other dynamic pruning method such as FBS. For each sample, a series of discrete action probability distributions $\{\mathbf{p}_l \in \mathcal{R}^m | l = 1, \cdots, L\}$ are produced as the testing sample passes through our model. We concatenate together the probability values of each layer to form the "*decision feature*" of this sample. We use UMAP (McInnes, Healy, and Melville 2018) to visualize the decision features produced by our method. Compared to FBS, we also use UMAP to visualize the concatenation of all layer channel modulation values produced by its decision unit.

Figure 3 shows the comparison result. Each point represents a sample, and colors represent different categories. In the embedding space, the decision features corresponding to the semantically close categories are closer, indicating that the distances of the decision features are related to semantic similarity. Compared to FBS, our method results in more distinct and diverse channel-wise modulation values for each sample. This demonstrates that FBS may not perform *real* dynamic pruning, since most of channel-wise gating values converge to a common channel compression scheme. This result validate the necessity of our discrete channel selection design.

Adversarial Example Detection

In this section, we will show how to use the dynamic pruning network to detect adversarial examples. The proposed dynamic pruning network introduces a decision unit to output discrete actions in each layer. This way of discretizing the internal features can help to understand the model's functional behavior and further analyze the model's predictions. For the adversarial sample, we consider that the decision paths within the network are significantly different from the normal sample. This difference can be used as a representative feature to distinguish between normal and adversarial samples.

Table 4: Adversarial sample detection AUROC(%) for different methods. Higher is better.

Dataset	Attack	Detection AUROC (%)				Our Detection AUROC (%)		
		KD + PU	LID	MAHA	FBS	Adaboost	RF	GBDT
CIFAR10	FGSM	81.21	99.71	99.92	100.00	100.00	100.00	**100.00**
	PGD	82.28	96.39	99.59	99.75	**99.97**	99.81	99.94
	DeepFool	81.07	88.47	**91.53**	81.16	88.43	84.27	91.44
	CW	55.93	82.93	95.85	97.49	99.14	97.67	**99.27**
CIFAR100	FGSM	89.90	89.27	99.77	100.00	100.00	**100.00**	99.99
	PGD	83.67	85.19	96.72	99.14	99.65	99.21	**99.65**
	DeepFool	80.22	64.80	**83.93**	74.31	78.42	72.22	81.03
	CW	77.37	75.35	91.65	96.62	98.60	97.19	**98.76**
SVHN	FGSM	82.67	95.72	99.63	99.95	**99.96**	99.85	99.95
	PGD	66.19	87.41	97.14	98.99	98.82	97.53	**99.17**
	DeepFool	89.71	88.81	**95.46**	90.34	86.39	83.74	92.21
	CW	76.57	85.66	92.13	96.35	96.70	93.66	**97.66**

Figure 4 shows how we use UMAP to visualize the decision features of the normal and adversarial examples. We use the dynamically pruned ResNet34 for feature extraction. All colored dots represent normal examples. Different colors represent different categories. The gray dots represent all the adversarial samples. It can be seen that there is a clear separation between the adversarial sample and the normal sample in the embedding space. The decision features from FGSM (Goodfellow, Shlens, and Szegedy 2014) adversarial samples have large distances from all normal samples, and for some more powerful attack methods such as DeepFool (Moosavi-Dezfooli, Fawzi, and Frossard 2016) and ℓ_2-version C&W attack (Carlini and Wagner 2017), there are overlaps between decision features. This can explain why the FGSM attack is easier to be detected in the following experiment section.

Based on the above observations, we propose a novel approach to conduct adversarial sample detection as a binary classification task. For a testing sample, we collect the layer-wise decision probability vectors $\{\mathbf{p}_1, \mathbf{p}_1, \cdots, \mathbf{p}_L\}$ and then concatenate them together to form the decision feature $\mathbf{P} = \text{concat}(\mathbf{p}_1, \mathbf{p}_1, \cdots, \mathbf{p}_L)$. A variety of binary classifiers can then be trained on these features as adversarial example detectors. We will further show the experimental setup and result details in the following section.

Experiments

In this section, we present the adversarial sample detection results based on the layerwise decision feature discussed in the previous section. Similar to the experimental setups in (Lee et al. 2018), we train a ResNet-34 model for classifying CIFAR10, CIFAR100, and SVHN datasets. Then the adversarial samples are generated with these four attacking methods, including FGSM, PGD (Kurakin, Goodfellow, and Bengio 2016), DeepFool and ℓ_2-version C&W attack.

For our method, we first use dynamic pruning technique to train the decision units and channel selection masks for each residual block. Then each sample's runtime layer-wise decision probability vectors are collected and concatenated as its decision feature. Unlike the previous procedure, the original model weights are not fine-tuned, since we want to examine the original model functional behavior without altering its decision process. This practice can thus be applied to any pre-trained models as an auxiliary network for analysis.

In the experiments, we utilize three conventional binary classifiers Adaboost, Random Forest (RF) and Gradient Boosting Decision Tree (GBDT) as detectors trained on decision features. We compare with three state-of-the-art logistic regression detectors, including kernel density (KD) and predictive uncertainty (PU) (Feinman et al. 2017), the local intrinsic dimensionality scores (LID) (Ma et al. 2018) and the Mahalanobis distance scores (MAHA) (Lee et al. 2018). Only 10% of randomly chosen test samples are used for training the binary classifiers and rest for evaluation.

Table 4 summarizes the detection Area Under ROC (AUROC) metric for each method. Based on the discriminative layerwise decision features, our method outperforms other detection methods across different attacking methods. Since the adversarial sample at the input is close to the normal sample, and there is a large deviation at the prediction end, it can be inferred that its propagation behavior is different from the normal sample. The proposed method can further amplify this difference due to the characteristics of its layerwise discrete decisions, which is more conducive to detecting the adversarial sample in the feature space.

Conclusion

In this paper, we propose a novel dynamic network pruning model with interpretable layerwise channel selection. We design a new decision head for each layer weight channel selections, where the action head and channel selection masks are jointly trained in a differentiable manner. By means of the runtime layerwise decision features generated by the our model, we propose a novel adversarial sample detection algorithm. Experiments show our method can achieve better pruning results and significantly improve model robustness against adversarial attacks.

Acknowledgements. This work was supported by the National Key Research and Development Program of China (No. 2017YFA0700904), NSFC Projects (Nos. 61620106010, 61621136008, 61571261, 61836014), Beijing Academy of Artificial Intelligence (BAAI), the JP Morgan Faculty Research Program and Huawei Research Program in collaboration with the Shield Lab of Huawei 2012 Research Institute.

References

Carlini, N., and Wagner, D. 2017. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, 3–14. ACM.

Feinman, R.; Curtin, R. R.; Shintre, S.; and Gardner, A. B. 2017. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*.

Gao, X.; Zhao, Y.; Dudziak, L.; Mullins, R.; and Xu, C.-z. 2018. Dynamic channel pruning: Feature boosting and suppression. *arXiv preprint arXiv:1810.05331*.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2014. Explaining and harnessing adversarial examples. *arXiv:1412.6572*.

Gumbel, E. J. 1954. *Statistical theory of extreme values and some practical applications: a series of lectures*, volume 33. US Government Printing Office.

Han, S.; Mao, H.; and Dally, W. J. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*, 770–778.

He, Y.; Kang, G.; Dong, X.; Fu, Y.; and Yang, Y. 2018a. Soft filter pruning for accelerating deep convolutional neural networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2234–2240. AAAI Press.

He, Y.; Lin, J.; Liu, Z.; Wang, H.; Li, L.-J.; and Han, S. 2018b. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 784–800.

He, Y.; Zhang, X.; and Sun, J. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, 1389–1397.

Jang, E.; Gu, S.; and Poole, B. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Kurakin, A.; Goodfellow, I.; and Bengio, S. 2016. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*.

LeCun, Y.; Denker, J. S.; and Solla, S. A. 1990. Optimal brain damage. In *Advances in neural information processing systems*, 598–605.

Lee, K.; Lee, K.; Lee, H.; and Shin, J. 2018. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *Advances in Neural Information Processing Systems*, 7167–7177.

Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; and Graf, H. P. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*.

Lin, J.; Rao, Y.; Lu, J.; and Zhou, J. 2017. Runtime neural pruning. In *Advances in neural information processing systems*, 2181–2191.

Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; and Zhang, C. 2017. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, 2736–2744.

Liu, Z.; Sun, M.; Zhou, T.; Huang, G.; and Darrell, T. 2019. Rethinking the value of network pruning. In *ICLR*.

Luo, J.-H., and Wu, J. 2018. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *arXiv preprint arXiv:1805.08941*.

Luo, J.-H.; Wu, J.; and Lin, W. 2017. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, 5058–5066.

Ma, X.; Li, B.; Wang, Y.; Erfani, S. M.; Wijewickrema, S.; Schoenebeck, G.; Song, D.; Houle, M. E.; and Bailey, J. 2018. Characterizing adversarial subspaces using local intrinsic dimensionality. *arXiv preprint arXiv:1801.02613*.

Maddison, C. J.; Mnih, A.; and Teh, Y. W. 2016. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*.

McInnes, L.; Healy, J.; and Melville, J. 2018. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *ArXiv e-prints*.

Metzen, J. H.; Genewein, T.; Fischer, V.; and Bischoff, B. 2017. On detecting adversarial perturbations. *arXiv preprint arXiv:1702.04267*.

Moosavi-Dezfooli, S.-M.; Fawzi, A.; and Frossard, P. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2574–2582.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *Nature* 550(7676):354–359.

Sutton, R. S.; Barto, A. G.; et al. 1998. *Introduction to reinforcement learning*, volume 1. MIT press Cambridge.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2013. Intriguing properties of neural networks. *arXiv:1312.6199*.

Tramèr, F.; Kurakin, A.; Papernot, N.; Goodfellow, I.; Boneh, D.; and McDaniel, P. 2017. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3-4):229–256.