Efficient and Robust Convolutional Neural Networks via Channel Prioritization and Path Ensemble

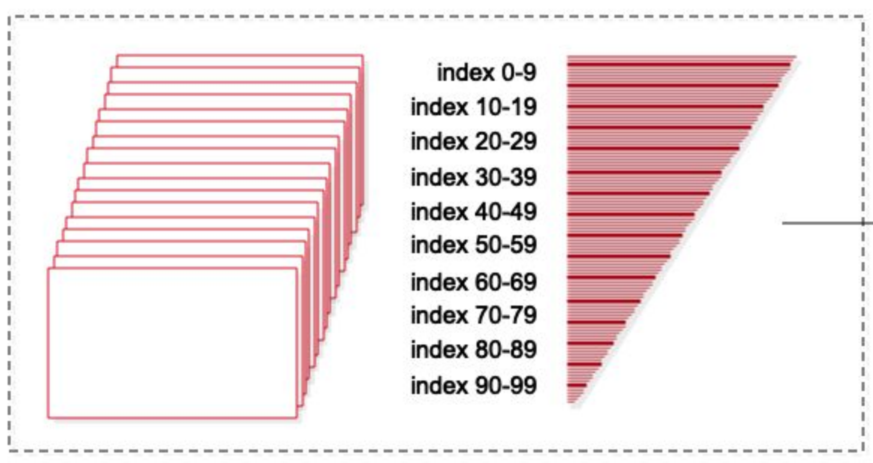
Chun-Min Chang[1], Chia-Ching Lin[2], Kuan-Ta Chen[1]   [1]Academia Sinica, [2]National Taiwan University

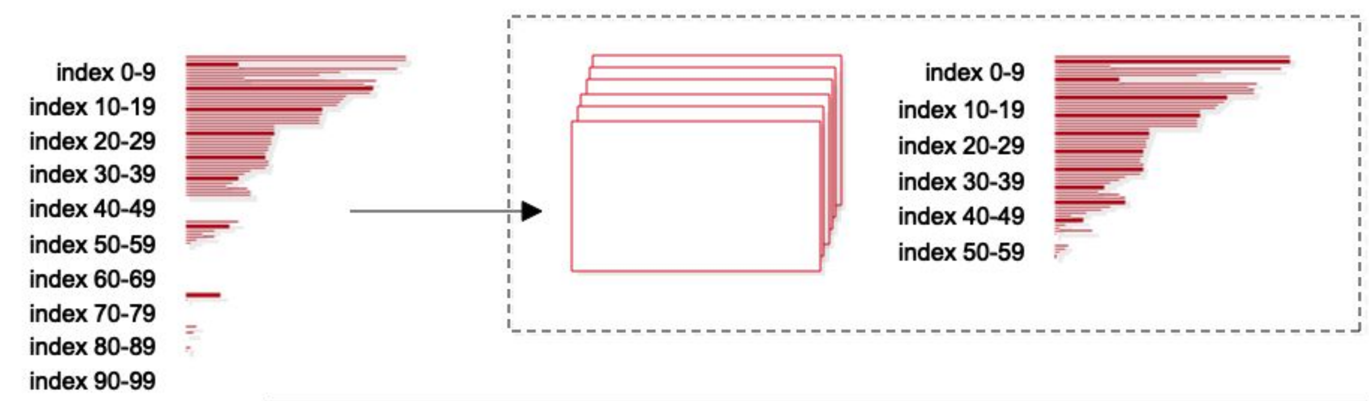
Overview

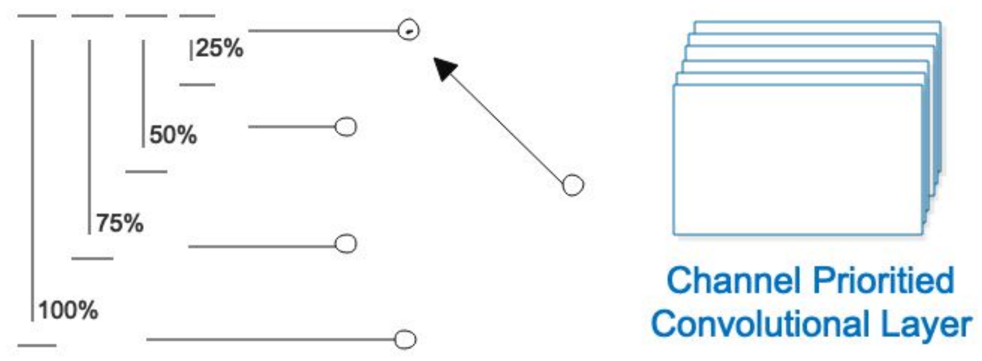
- Deploying modern neural networks on resource-limited platforms is often impeded by large model size and slow inference time
- A static model is hard to meet different device/application requirements such as power consumption, accuracy, and latency
- We propose a generalizable approach that search for an efficient architecture by pruning and achieve dynamically trade offs between performance and computation costs by switching between sub-paths in the pruned network

Approach

- We sparsify and prioritize channels by using scaling factors of batch normalization as the indicator of the channel's importance
- Once prioritizing channels in decreasing order, a sub-path at (100-p)% utilization can be built by pruning the aftermost p% channels
- Several sub-networks at different utilization levels are jointly trained to obtain the adaptive property







Pre-training stage

1. Monotonically decreasing initialization on scaling factors
2. Train with sparsity and monotonicity-induced penalties
3. Prune channels by a global threshold

Fine-tuning stage

4. Fix all parameters in BN layers
5. Define and sum up the losses at different utilization levels
6. Fine-tune the pruned network with the aggregated loss

- $$L_{obj} = Loss + \lambda_s \sum_{k,l} |\gamma_l^{(k)}| + \lambda_m \sum_{k,l} L_{m,l}^{(k)}$$

Monotonically decreasing initialization

$$\gamma_l^{(k)} = 1 - \frac{k-1}{N_l}, \quad k = 1, \ldots, N_l$$

Monotonicity-induced penalty

$$L_{m,l}^{(k)} = \begin{cases} \gamma_l^{(k+1)} - \gamma_l^{(k)} & , \text{ if } \gamma_l^{(k+1)} > \gamma_l^{(k)} \\ 0 & , \text{ otherwise} \end{cases}$$

Sparsity penalty

$$\sum_{k,l} |\gamma_l^{(k)}|$$

p: utilization level, the percentage of used channels

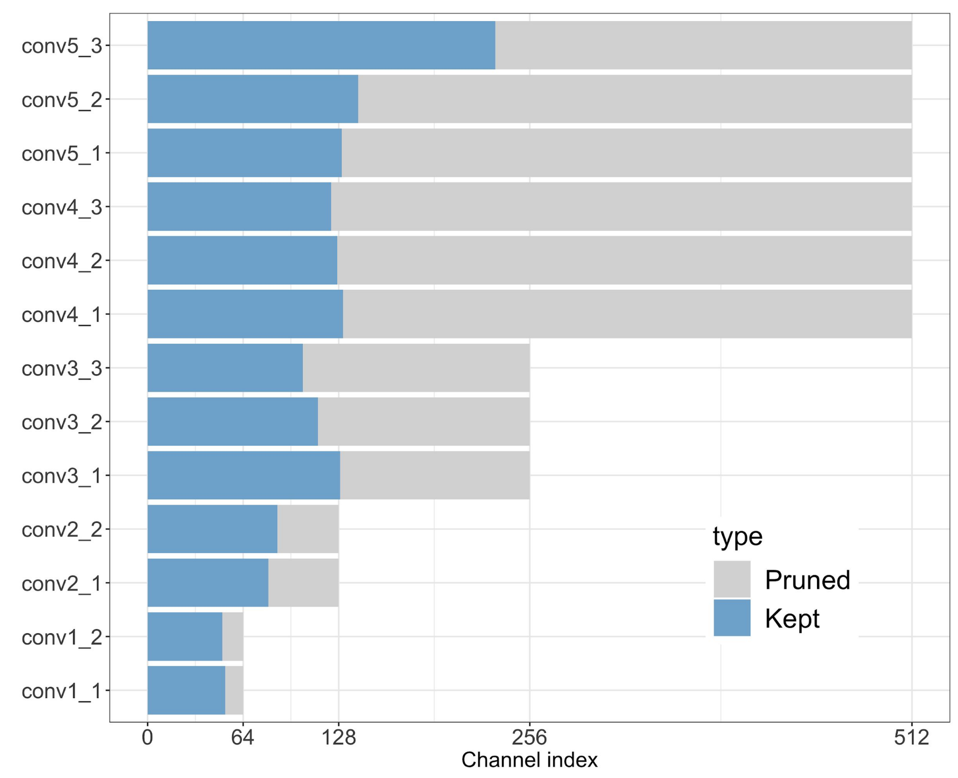
- $$L_{obj} = \sum_{p \in P} \alpha_p \times Loss_p$$

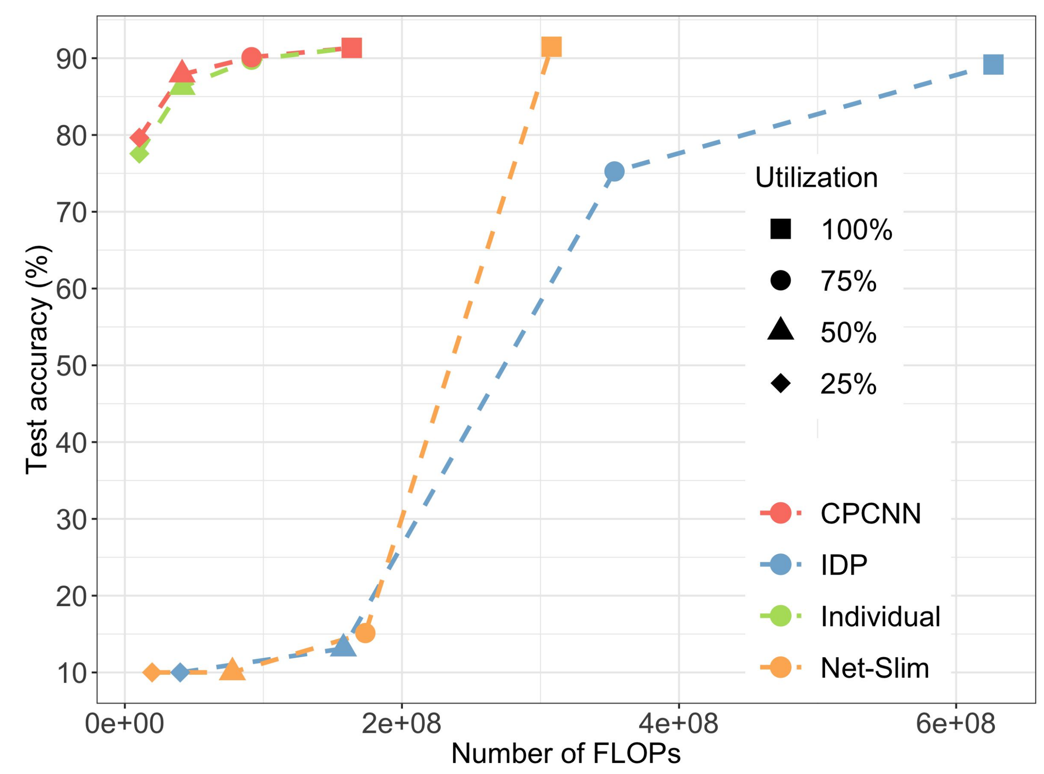
Experiment Results

Evaluation of Model Compression and Dynamic Inference Performance

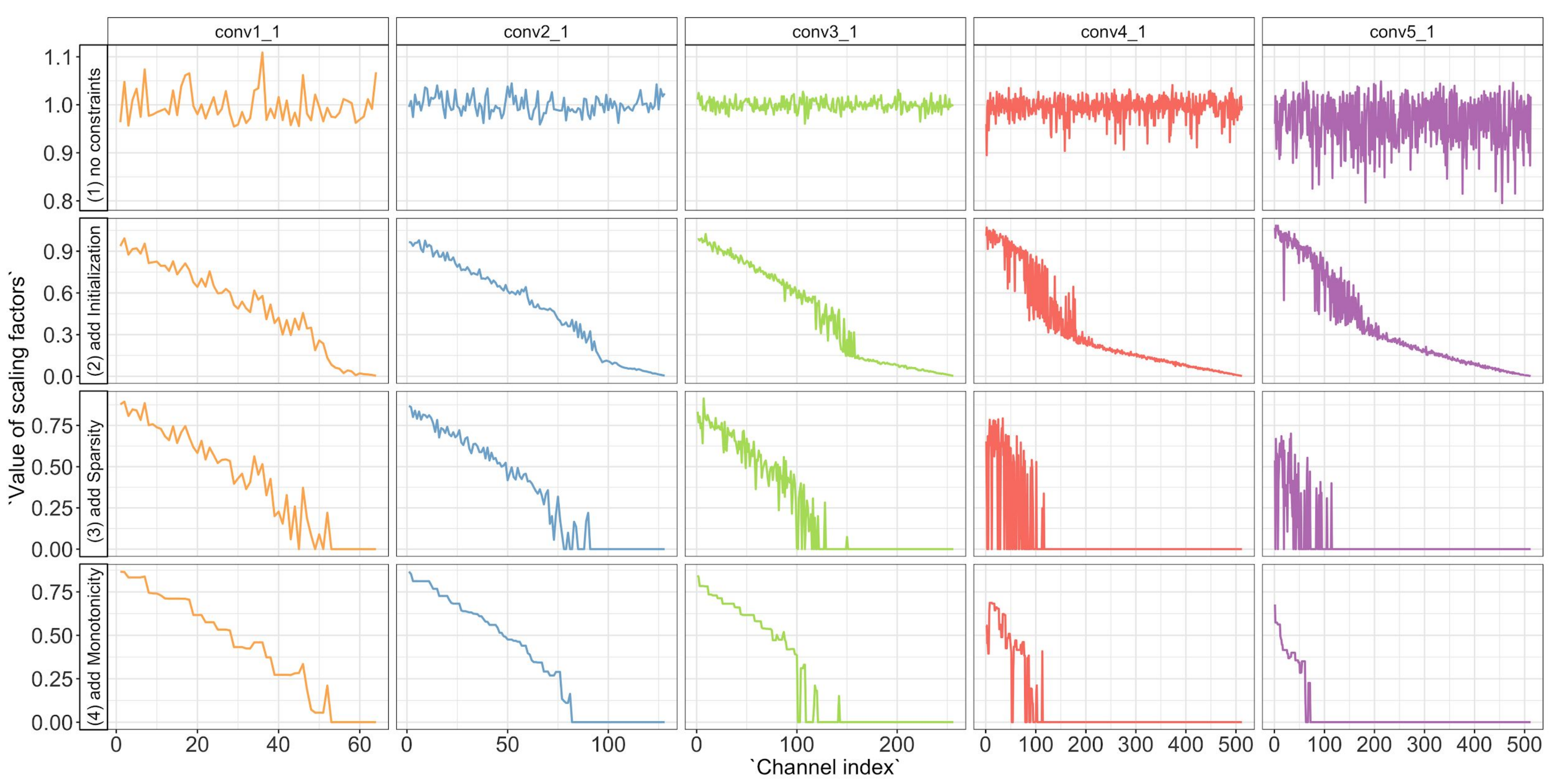
Model	Accu (%)	Param (M)	%	FLOP (M)	%
VGG-16	91.62	15.0	100.0	626	100.0
[4]†	91.49	1.40	9.4	308	49.1
CP (100%)	91.43	1.61	10.7	163	26.1
CP (75%)	90.13	0.92	6.2	91.5	14.6
CP (50%)	87.88	0.43	2.9	41.3	6.6
CP (25%)	79.64	0.13	0.8	10.4	1.7

Model	FLOPs at 100%	Accuracy at utilization level 100%	75%	50%	25%
[4]	3.08×10^8	91.49	15.13	10.0	10.0
IDP-linear	6.26×10^8	89.18	75.25	13.11	10.0
CPCNN	1.63×10^8	91.34	90.13	87.88	79.64
Individual‡	1.63×10^8	91.35	89.80	86.24	77.57





Ablation Study



Path Ensemble as Adversarial Defense

Attack	Test set accuracy at utilization 100%	75%	50%	25%	Path Ensemble
FSGM, $\epsilon = 0.5$	77.9	80.3	79.5	59.1	**81.3**
FSGM, $\epsilon = 1.0$	64.5	69.2	73.3	57.4	**73.3**
FSGM, $\epsilon = 2.0$	46.4	51.8	**60.1**	53.8	58.8
C&W, $k = 0.0$	10.5	15.1	15.8	**16.0**	15.7
C&W, $k = 2.0$	14.7	17.8	**18.7**	18.2	18.4
C&W, $k = 5.0$	24.2	25.6	25.8	24.1	**26.2**
MadryEtAl, $\epsilon = 0.5$	76.5	80.0	79.5	59.1	**81.6**
MadryEtAl, $\epsilon = 1.0$	53.9	63.6	**71.5**	57.3	70.0
MadryEtAl, $\epsilon = 2.0$	18.1	28.8	**51.1**	53.3	45.0
DeepFool	6.9	63.4	69.9	54.5	**72.6**
MomentumIter, $\epsilon = 0.5$	74.1	78.6	78.9	58.9	**80.5**
MomentumIter, $\epsilon = 1.0$	51.3	61.7	**70.3**	56.8	68.1
MomentumIter, $\epsilon = 2.0$	19.5	28.9	49.2	**52.6**	43.8
Overall	41.4	51.1	57.2	47.8	52.7

Conclusions

- To the best of our knowledge, it is the first method that retakes the priority control on channels in CNNs, enabling network pruning and dynamic inference to trade-off between various resource and performance requirements
- Experiments demonstrated that controlling channel priority during training will not notably degrade model performance
- As a side effect of path ensemble, the robustness against adversarial attacks can be improved by a significant margin without any computation or memory overhead