

# Pruning Filters In Convolution Neural Network

Vincent Martineau

Department of Computer Science and Software Engineering, Université Laval



## Introduction

We explore how reducing network expressivity can affect performance in Convolution Neural Network (CNN). The idea is to take a network that can handle a more complex task and remove filters that are the least important for the new task.

### Motivations :

- **Reduce** network size and execution time.
- **Run** network on less demanding hardware with similar accuracy.

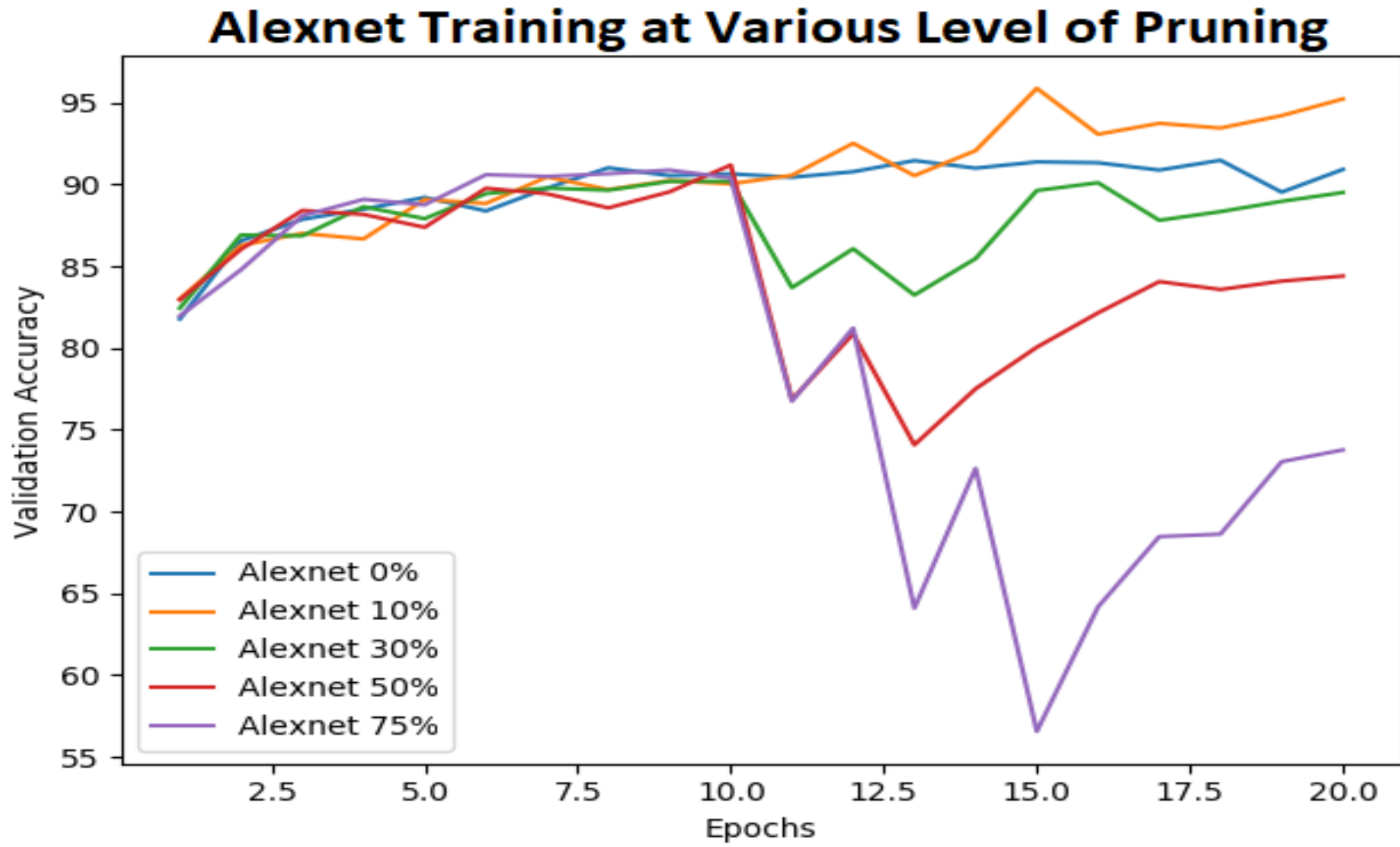
### Related work :

- P.Molchanov et al. (2017) : Pruning Convolutional Neural Networks for Resource Efficient Inference.

### Goals :

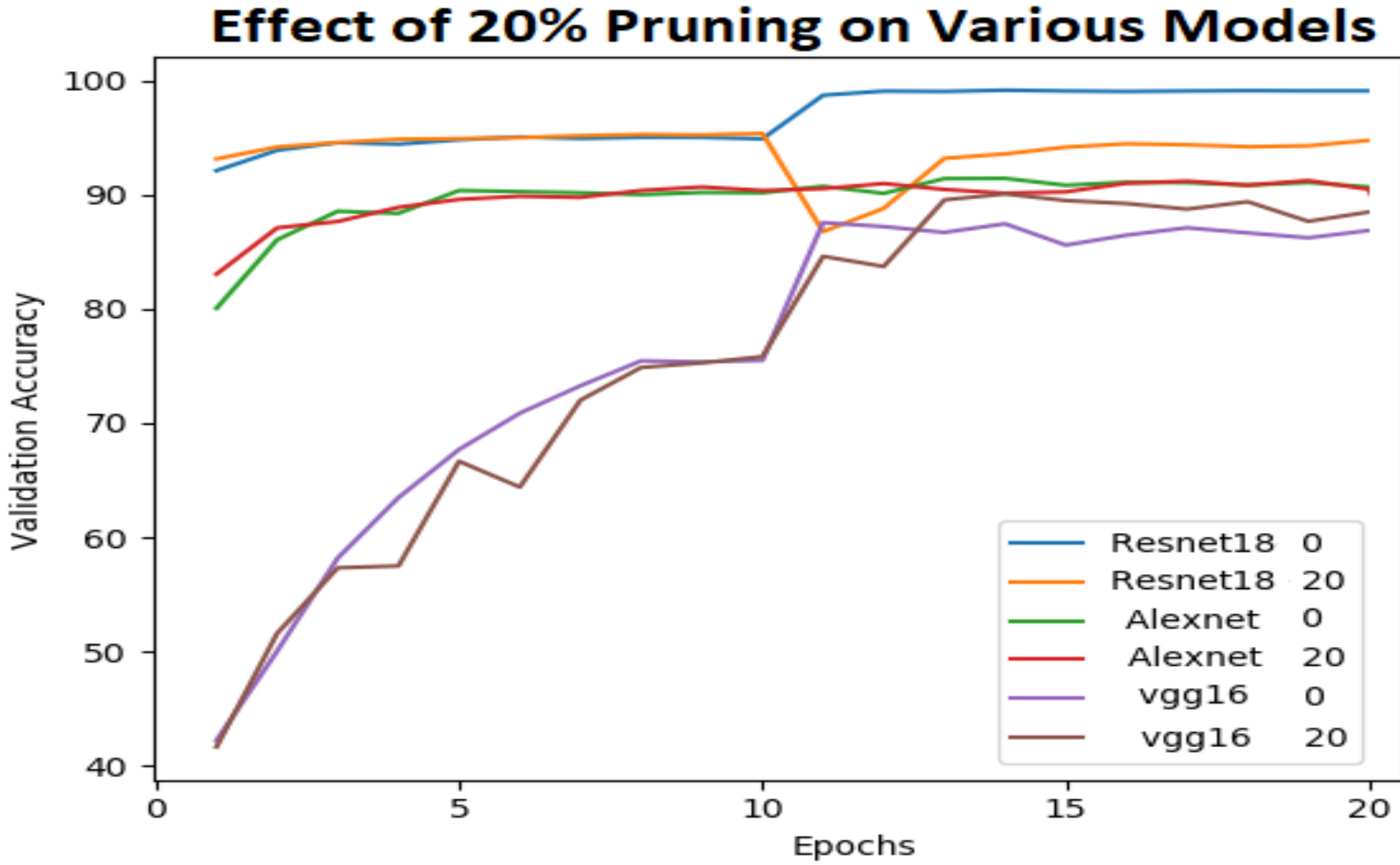
- Reduce training time to produce sufficient network.
- Provide a module that could handle multiple models.
- Explore the effect of pruning for speed and size.

## Comparing Various Level of Pruning



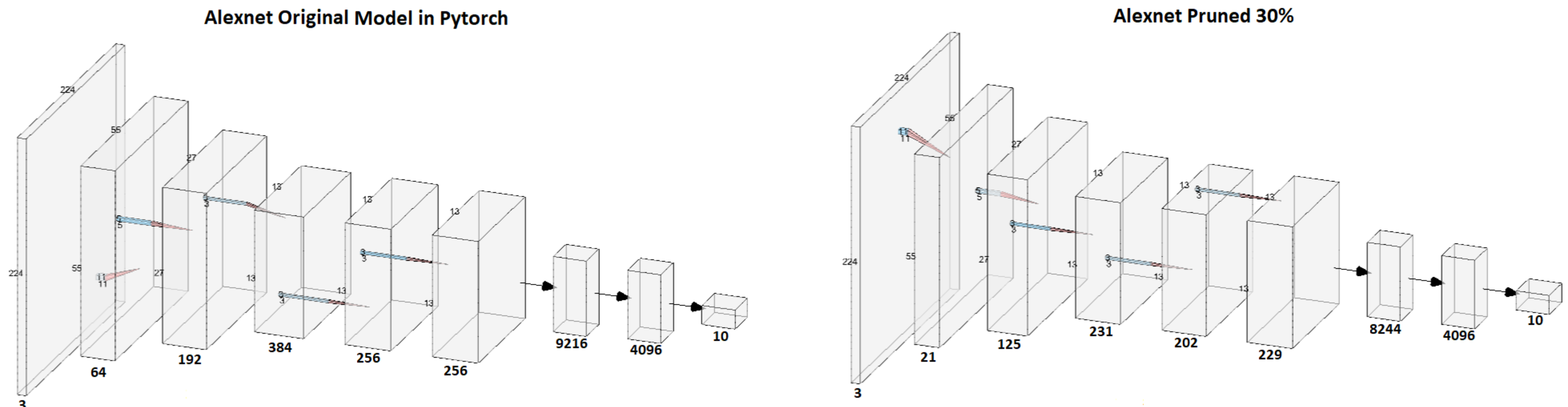
This graph compare various level of pruning. Each level of pruning is made on two iteration made after 10 epochs of training and 3 epochs or retraining. Pretrained weight were used to see the impact on transfer learning.

## Comparing Pruning on Multiple Models



This graph explore the effect of pruning on different models. Each model shows the effect of pruning 20% of the convolution filters in one step after 10 epochs of training.

## Example of Network Reduction



Comparing the effect of pruning on AlexNet. Left is the original model provided by pytorch. On the right is alex net pruned 30%. In the case of network like Alexnet there is an important reduction of parameters based on the reduction of the fully connected layer. Their is also an important reduction in the first layers.

## Algorithm

- Pretrain network with full paramters
- **Prepare pruning**
  - Convert model to ONNX
  - Extract execution graph
  - Determine which layer can be pruned
- **Prune network**
  - Find number of filter to prune on iteration
  - Wort filter based on activation mean
  - Remove filters
  - Apply pruning effect to next layers
  - Reset optimizer
- Finalize training

## Settings

### Common :

- **Optimizer** : Stochastic Gradient Descent
- **Learning Rate** : 0.01
- **Momentum** : 0.0
- **Nesterov** : False
- **Batch Size** : 64
- **Use GPU** : Yes

### Pruning :

- **Pretrain Epoch** : 10
- **Retrain Epoch** : 3
- **Total nb. Epoch** : 20

Nb Iteration on Alexnet : 2

Nb Iteration on Model Compare : 1

## Performances

Comparing Alexnet Attributes after Pruning					
	0%	10%	30%	50%	75%
<b>FLOPs(G)</b>	0.815	0.665 (-18.4%)	0.445 (-45.3%)	0.283 (-65.3%)	0.133 (-83.7%)
<b>Params(M)</b>	57.0	54.8 (-3.86%)	47.3 (-17.0%)	37.6 (-34.0%)	27.0 (-52.6%)

## Observations

- Not all convolutional layer can be pruned. Pruning layer before a residual connection is dangerous because both side of the residual connection must have the same side.
- When pruning in a convolution layer it is important to propagate. So the next layer have the right input size. This apply to convolution, linear and batchnorm layers.
- The algorithm used tend prefer removing filters that are deeper in the model and it is not uncommon to try to prune all filter in a layer.
- When pruning it is important to reset optimizer.

## Conclusion

### Discussion :

- **Morphology** and **context** help predict useful embeddings.
- **The attention mechanism works** : depending on the task, the network will use either more the context or the morphology to generate an embedding.

### Future works :

- Apply the **attention mechanism on each character of the OOV word and each word of the context** instead of using the hidden state of the respective elements only.
- Test our attention model in **different languages** and on other NLP tasks, such as **machine translation**.