Latent Pruning for Compute Optimization in 3D Object Detection Frameworks

Mir Sayeed Mohammad*, Mizanur Rahaman Nayan*, Sujoy Mondal*, Uday Kamal
Georgia Institute of Technology
CS 7641
*Indicates Equal Contribution

Introduction

Our work aims to reduce the computation in a deep neural network model deployed in 3D object detection task by removing channels in different model layers and optimizing inference. Structured pruning methods, such as CoFi, combine coarse and fine-grained units with layer-wise distillation to enhance model compression and inference speed without significant accuracy loss (Yu et al.; Wang et al.). Layer and head pruning in Transformer models also effectively reduce model size while maintaining performance (Yu et al.). Regularization-based pruning methods like L1 regularization and growing regularization gradually drive unimportant weights to zero before pruning, balancing parameter reduction and accuracy (Yeh et al.; Wang et al., "Neural Pruning via Growing Regularization"). Importance-based pruning uses criteria like neuron importance score propagation (NISP) and Taylor expansion-based methods to iteratively prune less important neurons, retaining essential features and performance (Yu et al.; Molchanov et al.; Sunil and Salem; Liu et al.). These techniques collectively improve the efficiency and performance of neural networks by systematically reducing their complexity.

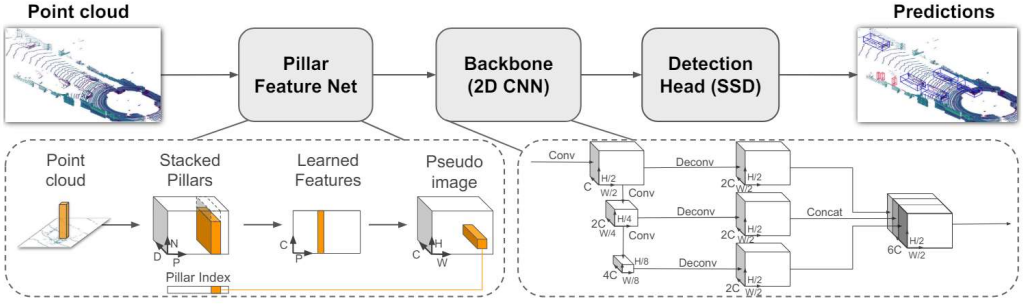
Problem Definition

LIDAR scanners are vital for autonomous systems, providing essential point cloud data. Processing this data incurs significant computational costs. Techniques like PointPillars (Lang, Alex H., et al.) convert point clouds into voxels, often leading to redundancy. Optimizing voxelized data processing enhances efficiency without sacrificing accuracy. Methods like Matryoshka Representation Learning prune feature channels, reducing computational load while maintaining performance. This project integrates pruning within PointPillars, using adaptive latent regularization to cut computation and memory needs, evaluating detection accuracy across various scenes and object classes.

Methods

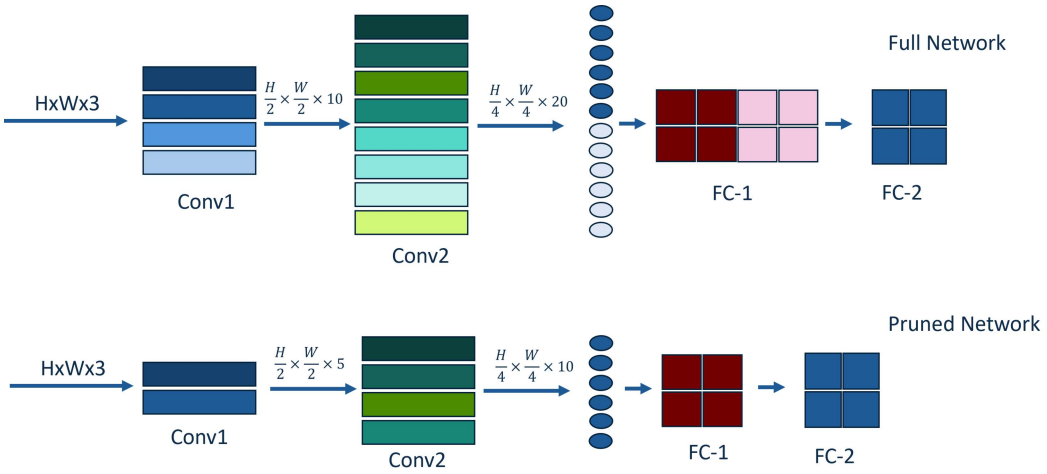
Pointcloud data is a stream of points reflecting LASER in the Cartesian coordinate system. For object detection, the space is converted to a grid structure called voxels. In the PointPillars architecture, points are filtered within a field of view, grouped into pillars in the x-y plane, and augmented to ensure each pillar contains the same number of points. The pillars then pass through an invariant dense layer, creating a fixed number of features, forming 2D pseudo images. Noise is added during preprocessing and pillar encoding to compensate for sensor uncertainty, with additional augmentation methods like rotation and flipping.

Object detection frameworks commonly use algorithms such as anchor generation, bounding box encoding, and non-max suppression. Models like PointNet, PointNet++, PV-RCNN, VoxelNet, and F-ConvNet are utilized for 3D object detection. These pipelines include modules like the encoder, backbone, and model head, with the dense backbone handling most feature processing and computation. Our goal is to develop a model-agnostic learning approach, ensuring the pruning method applies to all convolution/dense layer-based architectures.



PointPillars architecture

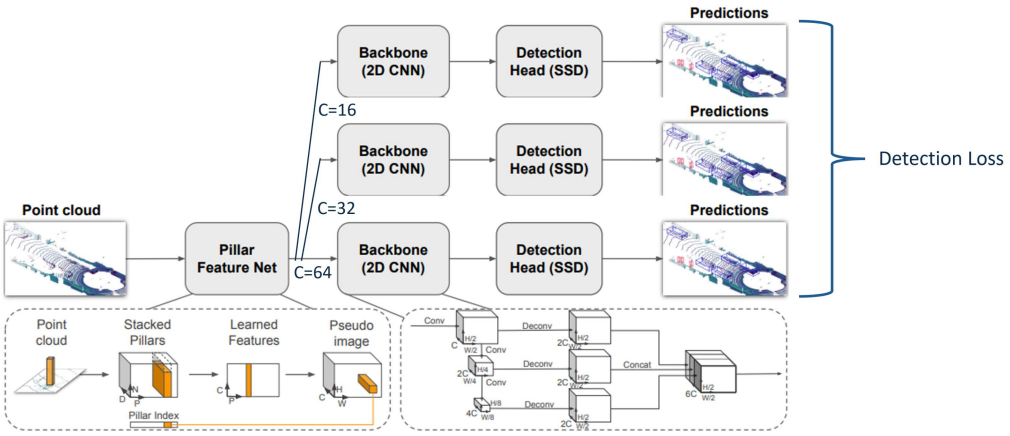
Our work optimizes model architecture by pruning parallel channels in each layer. Not all feature channels are necessary, and the latent space often contains redundant features. The pruning method regularizes channels, imposing higher penalties on higher channel outputs, encouraging the model to minimize unnecessary channels to zero output. During inference, a distilled smaller model retains only the non-zero parts of the parent model, improving efficiency while reducing computational requirements.



Proposed pruning method

Midterm Checkpoint

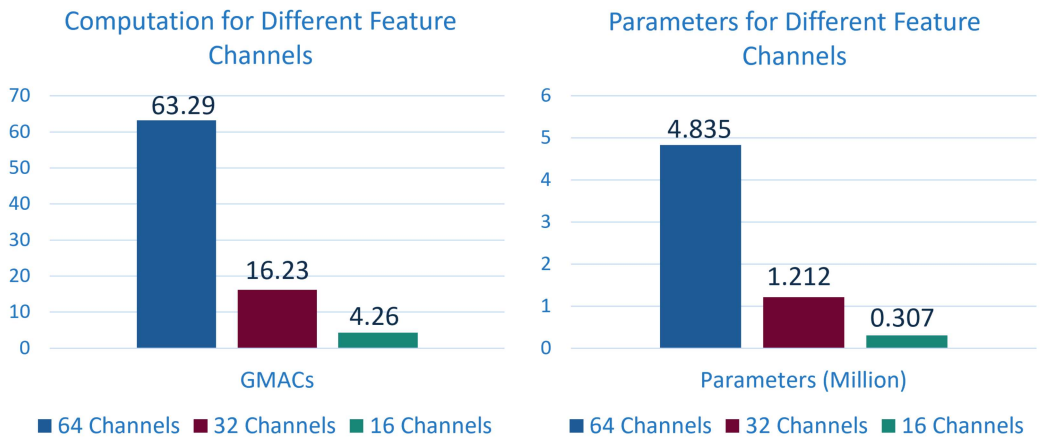
We jointly trained several model heads simultaneously utilizing different number of channels from the pillar feature extractor. The detection losses for each head are jointly optimized. The modified training scheme is provided below:



Implemented Matryoshka representation learning (Kusupati, Aditya, et al.)

The key motivation behind this implementation is that the point cloud feature extractor is common to all the model heads, and based on the desired level of computation, a different model head can be selected that uses more or less

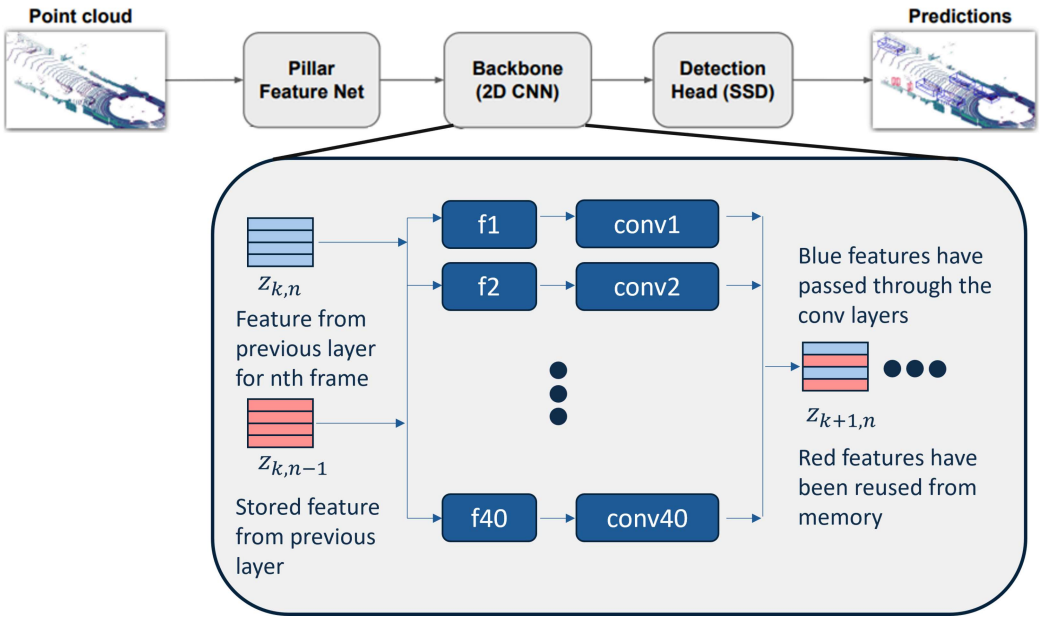
number of features. The computational advantage of the different model heads are the key motivation behind this scheme.



Reducing feature volume from pillar feature net to backbone by 75% can reduce host computation by ~93.2% and memory usage by ~93.6%.

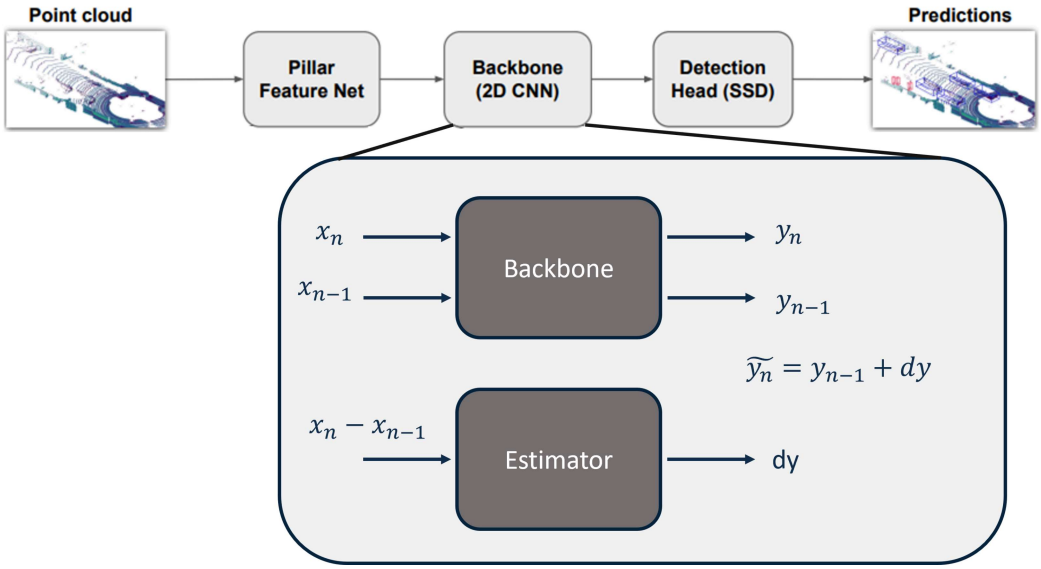
Final Checkpoint

In the second stage of our project, we take a look at reducing computation along the temporal axis of the video point cloud data. In this part, the key argument is that for very small time steps, the change of features in the spatial domain is also very small. So we can (1) Reuse some features calculated in the previous time step (2) Linearize some features calculated in the previous time step and estimate the future featuers. Based on this argument, we experiment with three separate models. (a) Feature Gating (b) Temporal Feature Linearization (c) Adaptive Feature Linearization. All these experiments aim to take advantage of the temporal redundancy of the data.



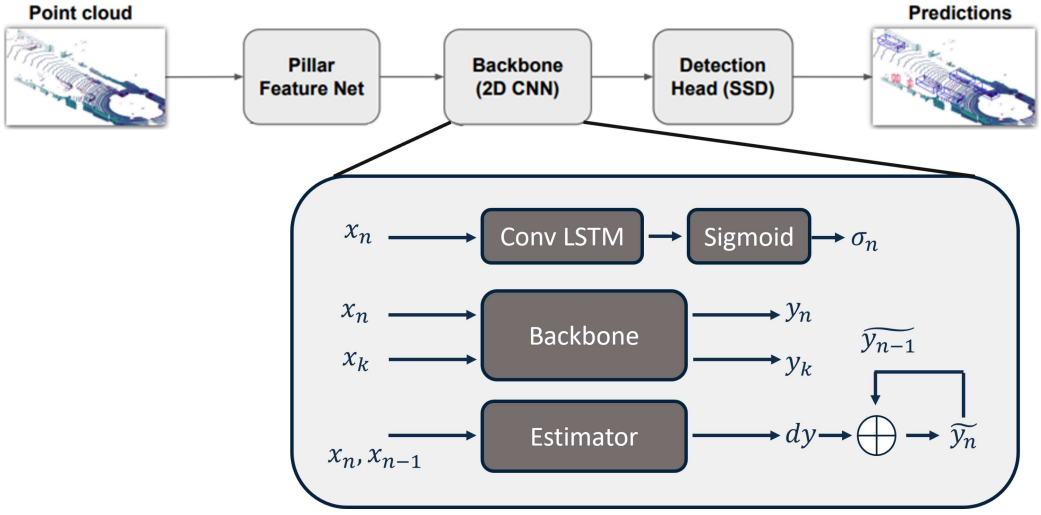
(a) Feature Gating

(a) NN models have many parallel operations, i.e. each output channel of a convolutional layer uses a separate filter kernel. Parallel computations can be replaced by reusing features from a previous time step. We introduce a temporal gating function that chooses whether to recompute new features or pass existing features from previous time-step to the next layer.



(b) Temporal Feature Linearization

(b) Instead of fully computing features in each time step, we try to estimate the outputs of the backbone with a smaller model. The small estimator model bypasses the backbone and estimates the output feature difference based on the input feature difference. Bypassing the backbone can hugely reduce the computation



(c) Adaptive Feature Linearization

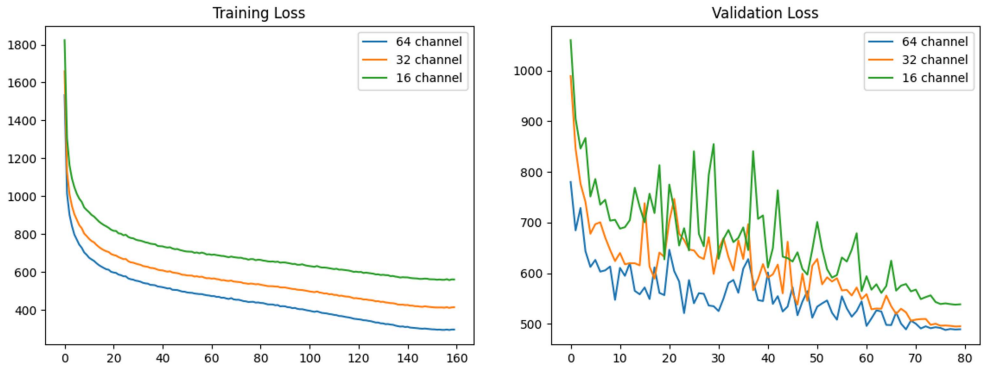
(c) The adaptive gating mechanism makes decision on when to start a new estimation sequence. The gate creates a geometric probability of when to switch based on the switching on previous time steps.

Results and Discussion

The goal is to reduce frame-by-frame computation in object detection while maintaining performance. We will evaluate performance vs. computation, aiming to minimize the curve's slope, meaning reduced computation with minimal performance loss. Key metrics: 2D AP, BEV AP, 3D AP, and MACs/FLOPs during inference.

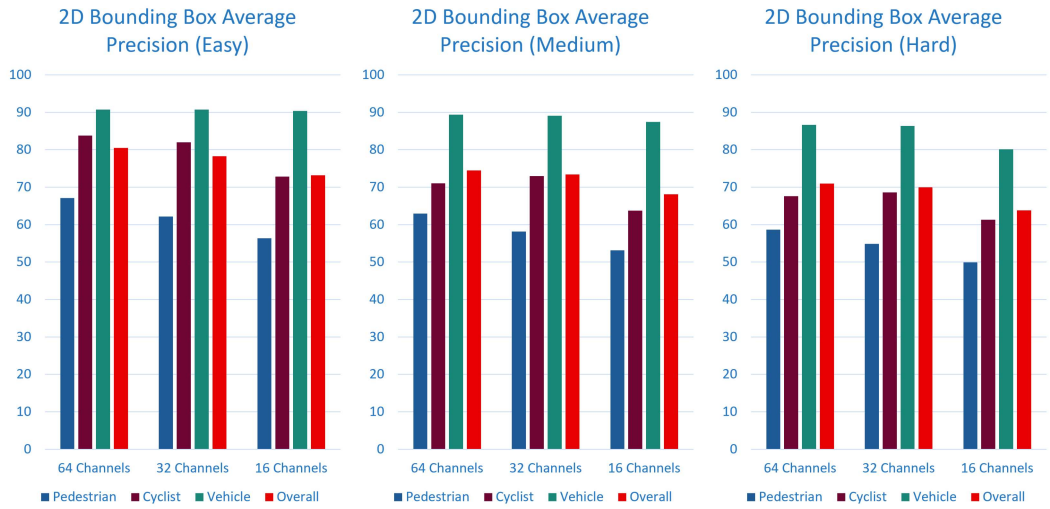
Midterm Checkpoint

The joint training scheme converges after 160 training epochs and are evaluated for performance.



From train and validation loss plots, we can see that lowering the number of channels causes the loss to saturate at a higher global minima, which is also reflected in the performance metrics.

One interesting key observation in the 2D bounding box average precision is that the detection of vehicle class objects are consistent for reduced feature channels, meaning that for easy/less cluttered scenes, vehicles can be easily detected at 93% less computation overhead - which is a signifiant achievement. The key problem is knowing when to switch between the different model heads.



2D bounding box average precision for easy-medium-hard splits for different feature channels

Visual detection results in an easy scene. We can see that vehicles (large objects) are mostly detected fine in easy scenes for any number of feature channels.

Visual results in a difficult scene. We can see that lowering the number of channels is often causing false detections in smaller objects.

Final Checkpoint

In the following table, we combine the results of the Matryoshka representation learning scheme as well as the temporal feature reuse methods that we experimented with.

Methods	2D Bounding Box AP	Average Orientation Similarity (AOS) AP	Birds Eye View (BEV) Bounding Box AP	3D Bounding Box AP	Average compute compared to baseline
Baseline (Full Model)	90.6297	90.6535	87.6656	71.3733	100%
Temporal Feature Reuse					
Feature Gating (without penalty)	84.2302	83.2848	72.6594	55.1423	~86.6%
Temporal Feature Linearization	65.3631	67.0945	41.1521	38.1171	55.4%
Feature Linearization (Adaptive)	69.3429	58.2475	54.7610	28.9403	~77.6%
Channel Pruning					
Matryoshka Representation Learning (Full)	91.2389	--	--	76.5421	100%

2D average precision, orientation similarity, birds eye view and 3D average precision results for all trained models.

Note that the channel pruning and the temporal feature linearizations have been conducted orthogonally, and combination of optimization in both axes will result in better performance of the entire joint architecture.

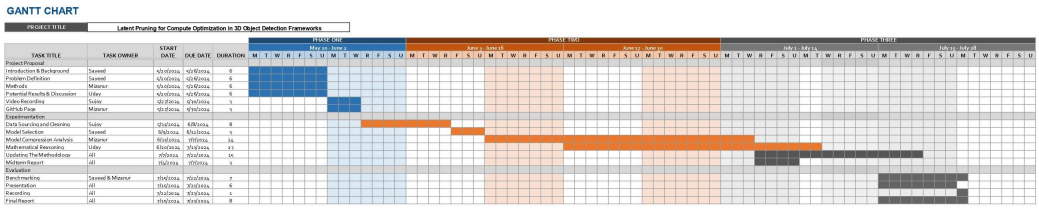
Discussion

In this project, we have explored several channel pruning methods, among which Matryoshka representation learning has provided the best results as reported here. Several other experiments are ongoing where we are trying to implement a parameter shared masking method for this which will allow us to dynamically choose the number of channels to be used during inference unlike here where the channel count is arbitrarily fixed during training. The disadvantage of Matryoshka learning is that it does not take advantage of the temporal nature of the data, hence the experiments with gating and temporal feature linearization. Temporal linearization methods discussed here are promising, but the performance is not there yet, leaving room for better linearization/estimation methods.

Future Steps

As future work, we plan to

1. Incorporate adaptive gating to Matryoshka representation
2. Implement Matryoshka model heads as a single feature-shared unit
3. Combine feature linearization and channel gating for better feature estimation
4. Experiment with LSTM feature predictor in place of feature linearizer
5. Use feature flow estimator similar to optical flow to better linearize the features along the time axis.



Gantt Chart for Workflow

References

1. Lang, Alex H., et al. "Pointpillars: Fast encoders for object detection from point clouds." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019.
2. Yu, Chenglin, et al. "Structured Pruning Learns Compact and Accurate Models." *arXiv*, 2022.
3. Wang, Jiaxiang, et al. "Structured Pruning for Deep Convolutional Neural Networks: A Survey." *arXiv*, 2023.

4. Yeh, Chih-Kuan, et al. "Deep Trim: Revisiting L1 Regularization for Connection Pruning of Deep Networks." *Papers with Code*, 2019.
5. Wang, Jiaxiang, et al. "Neural Pruning via Growing Regularization." *arXiv*, 2020.
6. Yu, Ritchie, et al. "NISP: Pruning Networks Using Neuron Importance Score Propagation." *arXiv*, 2017.
7. Molchanov, Pavlo, et al. "Importance Estimation for Neural Network Pruning." *arXiv*, 2019.
8. Sunil, Vadera, and Salem Ameen. "Methods for Pruning Deep Neural Networks." *arXiv*, 2020.
9. Liu, Zhuang, et al. "Rethinking the Value of Network Pruning." *arXiv*, 2019.
10. Kusupati, Aditya, et al. "Matryoshka representation learning." Advances in Neural Information Processing Systems 35 (2022): 30233-30249.

Video Presentation