Label-Efficient Steering Control with I-JEPA for Autonomous Driving

Venkat Kumar Laxmi Kanth Nemala
vn2263
MS Computer Engineering
NYU Tandon

Tejdeep Chippa
tc4263
MS Computer Engineering
NYU Tandon

Can you design a system for a downstream task of steering a self-driving car in CARLA simulator that is based on pre-training with I-JEPA and label-efficient fine-tuning?

Breakdown - What we had to do

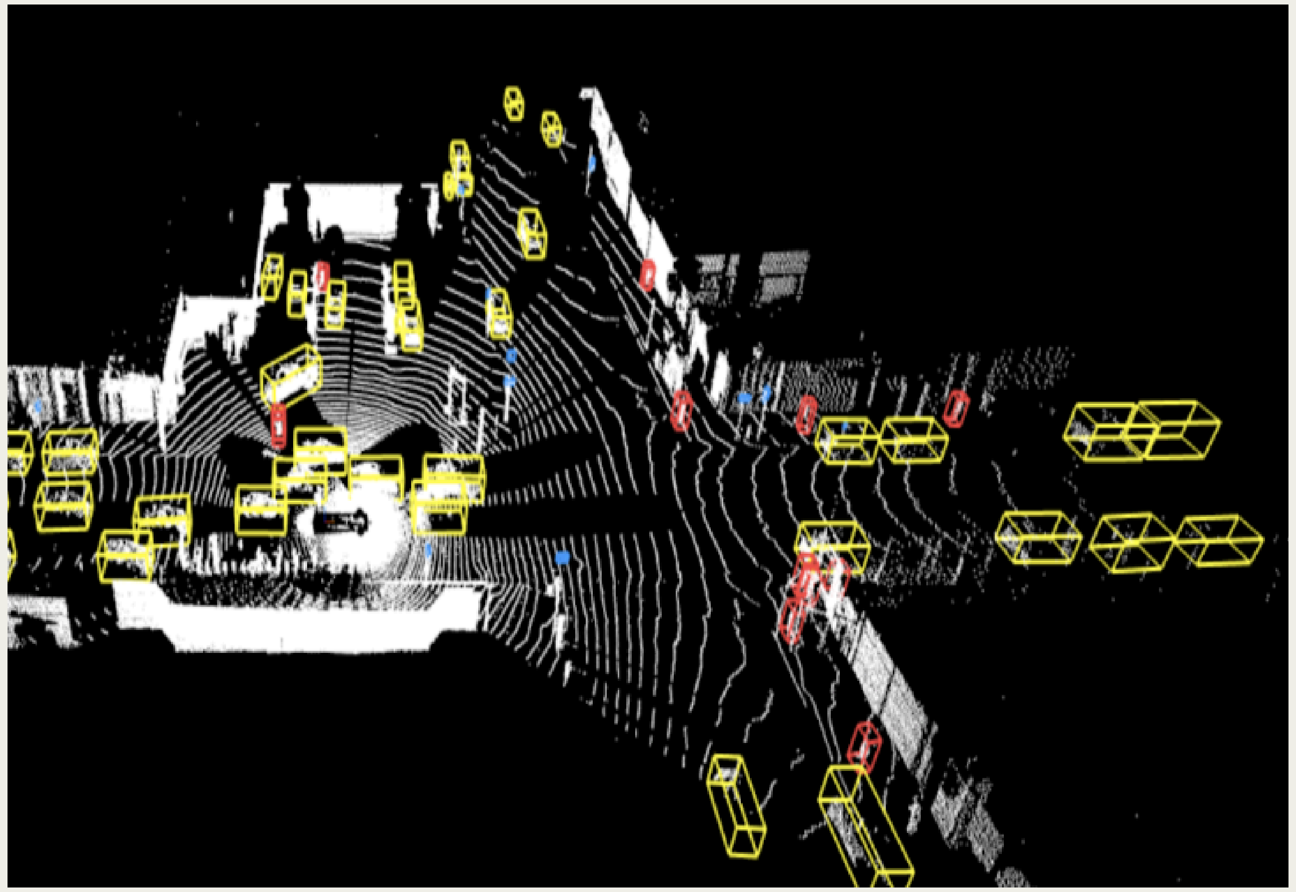
- Steering control is essential for safe autonomous driving
- Our Predict continuous steering angles from single front-view camera
- Use minimal labeled data for fine-tuning
- Use IJEPA an SSL based architecture to first pretrain a lot of images such that our method works even with minimal labeled data

DATASET DISCUSSION

We initially attempted to use large-scale real-world datasets like Waymo and KITTI, aiming to extract steering angles from their metadata.

We adopted the CARLA Steering Dataset for Self-Driving Cars, a high-quality simulated dataset comprising 84,000 images paired with precise steering angle labels.

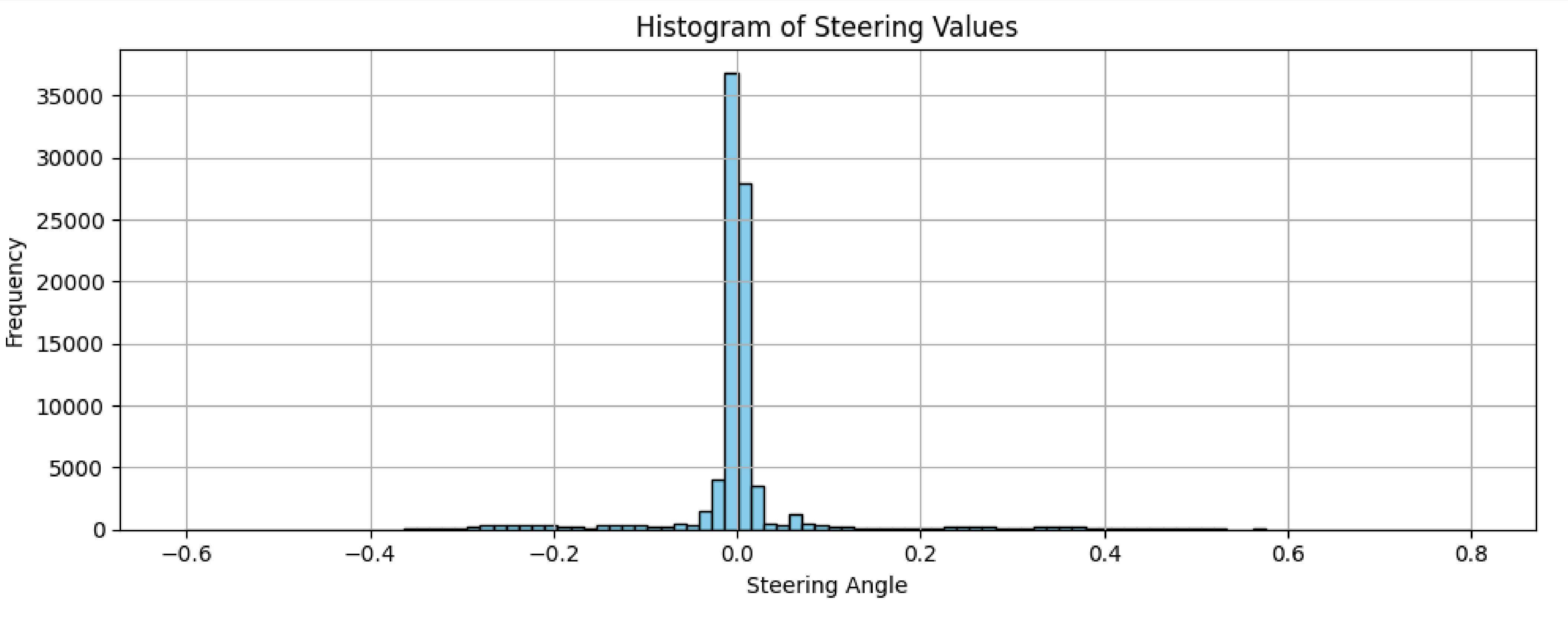
For our label-efficient fine-tuning, we used a curated subset of 10,000 samples, enabling targeted learning with minimal supervision.

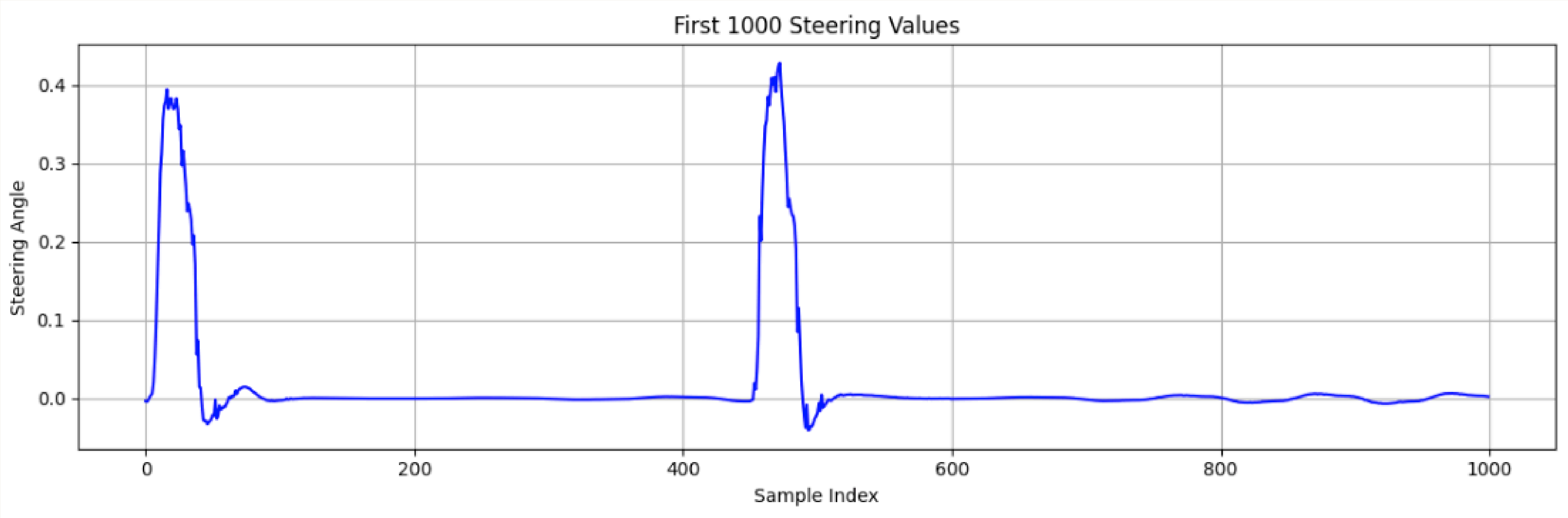


DATASET DISCUSSION

Data Statistics:

Total samples : 86491
Mean : 0.005479
Min value : -0.602661
Max value : 0.800000



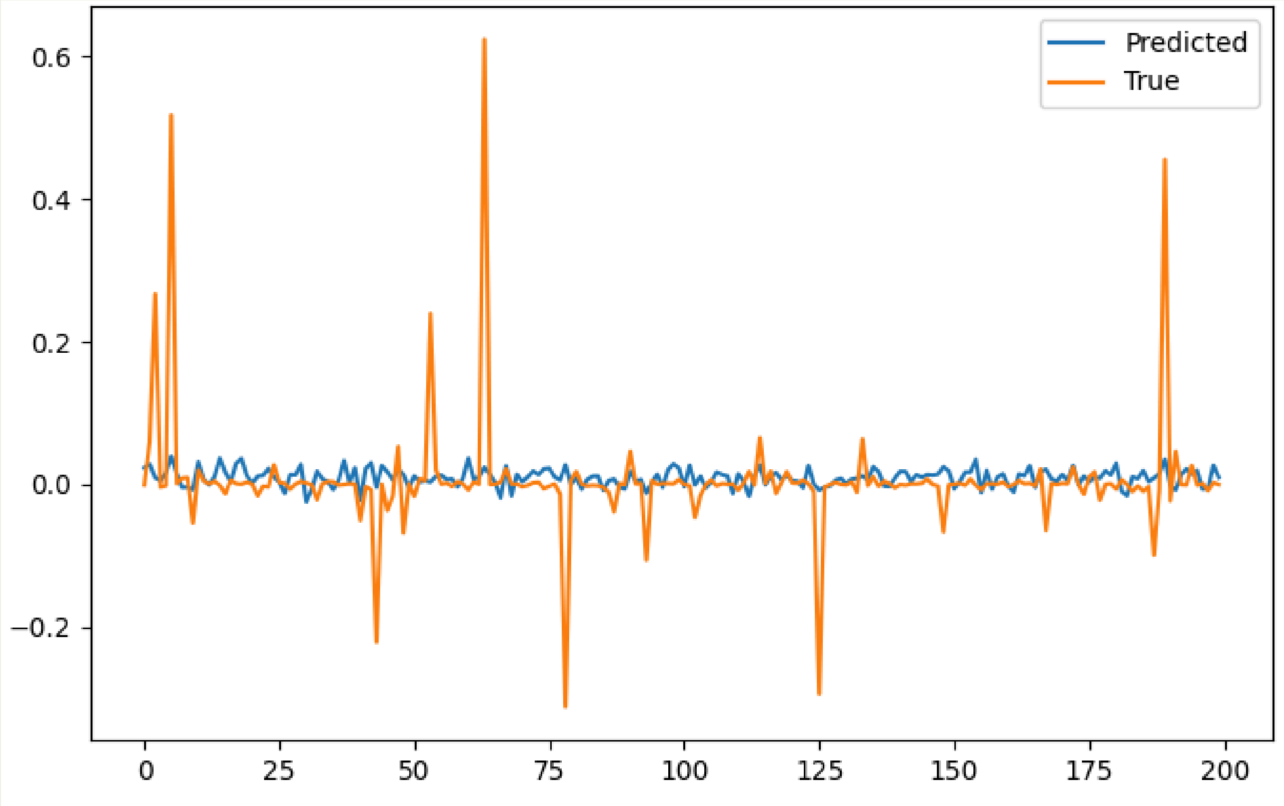


PREVIOUS METHODS

Without IJEPA - Supervised Learning

We tried out the supervised learning method first to figure out how it was performing.

- A classic convolutional neural network
- An advantage over the pretrained methods by using ROI cropped images
- Direct regression of steering angle
- Used MAE to error along with steering value scaling
- Scaling helps in better learning



Issues:

- Model struggled to fit large steering spikes
- Predictions mostly biased toward zero (straight line driving)
- Doesn’t work well even with low loss figures

INTRODUCING I-JEPA

Core Idea

- Traditional SSL (e.g., MAE) reconstructs raw pixels — sensitive to color, texture noise.
- I-JEPA instead predicts latent feature embeddings of masked regions from unmasked context.

Training Flow

1. Input: Full image ↠ masked spatial blocks are hidden.
2. Context Encoder: Processes only visible patches to build a context representation.
3. Predictor: Takes context and predicts target embeddings for masked regions.
4. Target Encoder (Frozen): Computes true embeddings from full image for supervision.

Loss Function

- Regression loss (MSE) between predicted embeddings and true target embeddings.
- No pixel-level loss, no contrastive loss.

Benefits

- Semantic abstraction: Model learns what is in the missing region, not how it looks.
- Robustness: Better generalization across datasets, distortions, downstream tasks.
- Efficiency: Faster convergence compared to reconstruction or contrastive methods.

IMPORTANT IDEAS IN IJEPA

Masked Latent Representation Prediction

- Predicts representations of masked image regions instead of pixel-level details
- Learns semantics rather than low-level reconstruction

No Pixel Reconstruction or Contrastive Loss

- Avoids the pitfalls of MAE and contrastive frameworks
- Leads to stable training and better sample efficiency

Efficient and Scalable Training

- Learns abstract scene-level understanding early
- Requires fewer training steps to converge on downstream tasks

Foundation for Multimodal Learning: ADL-JEPA

- Architecture naturally extends to audio, video, LiDAR, and text
- Key enabler for future multimodal perception in self-driving systems

ADLJEPA - A CLEVER EXTENSION

I-JEPA assumes that all context patches are equally important. ADL-JEPA adds an attention mechanism that dynamically selects important context for prediction.

Has an Adaptive Layer (ADL) that :

- Learns to selectively weight different visible patches.
- Suppresses irrelevant patches (e.g., sky, background) and emphasizes important ones (e.g., road, car edges).

Particularly useful when context has a lot of noise or unrelated patches, like in driving scenarios.

The Adaptive Layer is a simple addition with no massive overhead.

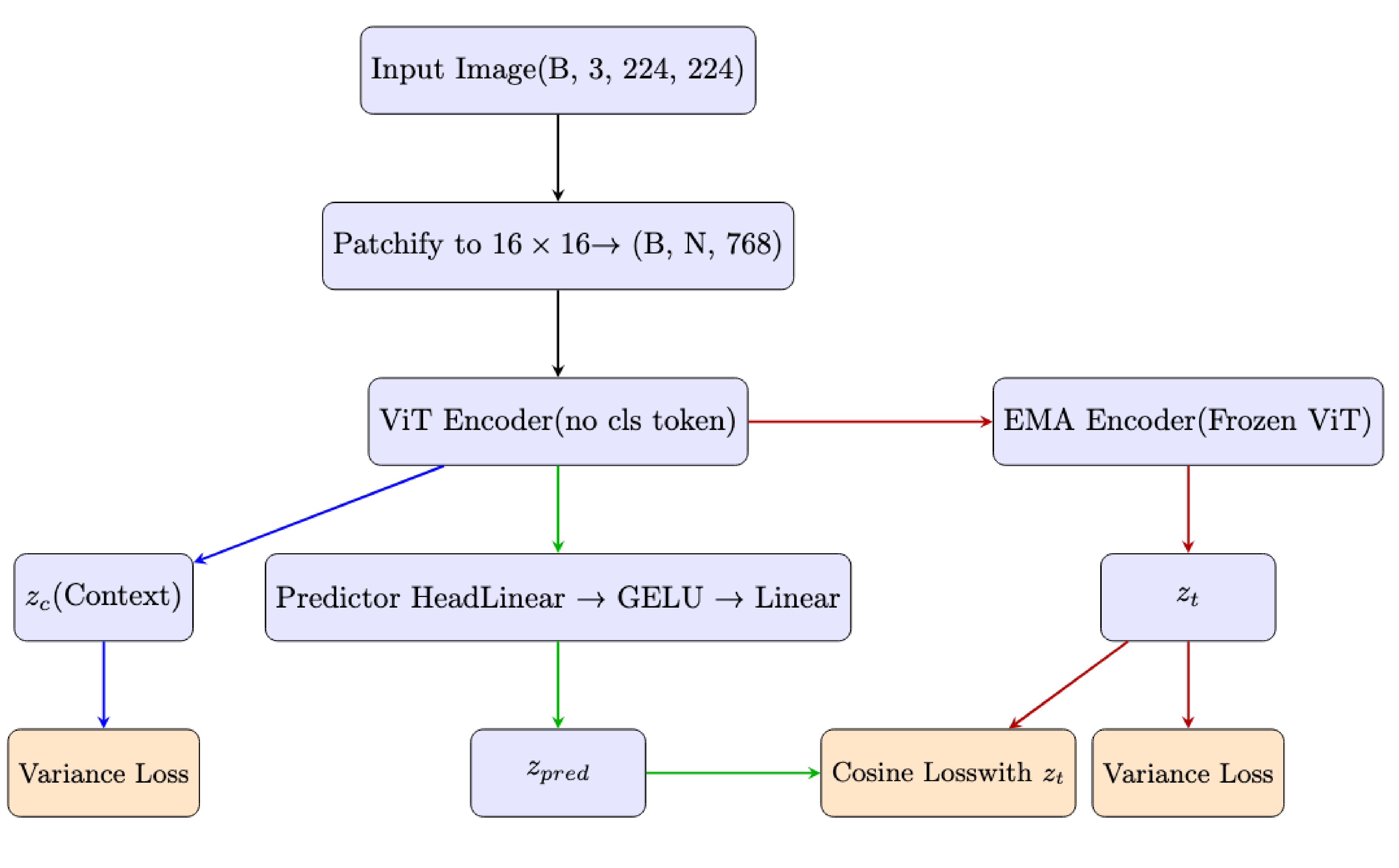
I-JEPA PRETRAINING ON STEERING DATA

Architecture:

- Backbone: Vision Transformer (vit_base_patch16_224)
- Predictor head: 2-layer MLP with GELU activation
- No decoder or reconstruction; operates in latent space

Pretraining Configuration

- Optimizer: AdamW
- Learning Rate: 1e-4
- Batch Size: 128
- Epochs: 100
- Patch Size: 16 × 16
- Masking Ratio: 50% patches masked
- Variance Loss Weight (λ_e): 25.0
- EMA Decay Rate: 0.996
- Input Image Size: 224 x 224
- Model: Vision Transformer backbone + MLP predictor
- Loss Functions: Cosine similarity loss + Variance regularization



I-JEPA PRETRAINING ON STEERING DATA

Dataset:

- pretraining performed on CARLA simulator steering dataset
- Resized all images to (224,224)

Masking Strategy:

- Patch-level random masking (50% of 16×16 patches masked per image)
- Encourages contextual prediction from surrounding visible patches

Regularization Techniques: To Prevent Representation Collapse

- Block dropout (implicitly through patch masking)
- EMA (Exponential Moving Average) of network weights as stable target
- Cosine similarity loss on masked patches
- Variance loss to prevent representation collapse across features

The I-JEPA pretraining converged with a final cosine loss of ~0.0002, learning rich masked patch representations for downstream steering control.

LABEL-EFFICIENT FINE-TUNING

Final Training Configuration

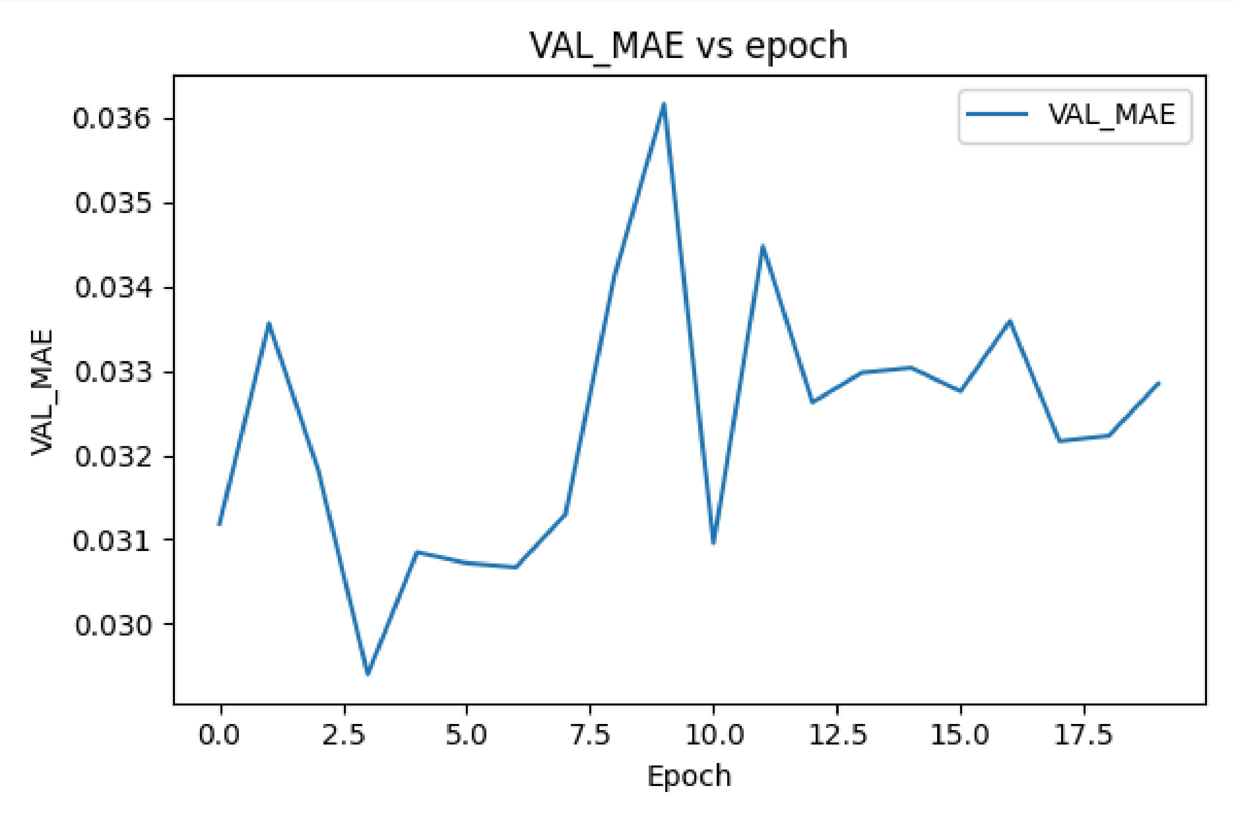
- Input Size: 224 x 224
- Optimizer: AdamW
- Learning Rate: 1e-4
- Batch Size: 64
- Epochs: 20
- Loss Function: Combined MSE + cubic loss for better handling of large steering errors
- Encoder: I-JEPA backbone frozen (only regression head trained)
- Dataset: ~10,000 images
- Data Split: 50% turning (spike) samples + 50% straight-line samples

LABEL-EFFICIENT FINE-TUNING

Our experiments and learning

First set of naive experiments –

Experiments with various percentages of dataset for label effecient finetuning



Learnings:

- Better to use a squared error function after scaling to emphasise more on the spikes that represent the actual turns.
- Data values of steering angles are very small in magnitude, hence using a scaling factor helps in having a better

LABEL-EFFICIENT FINE-TUNING

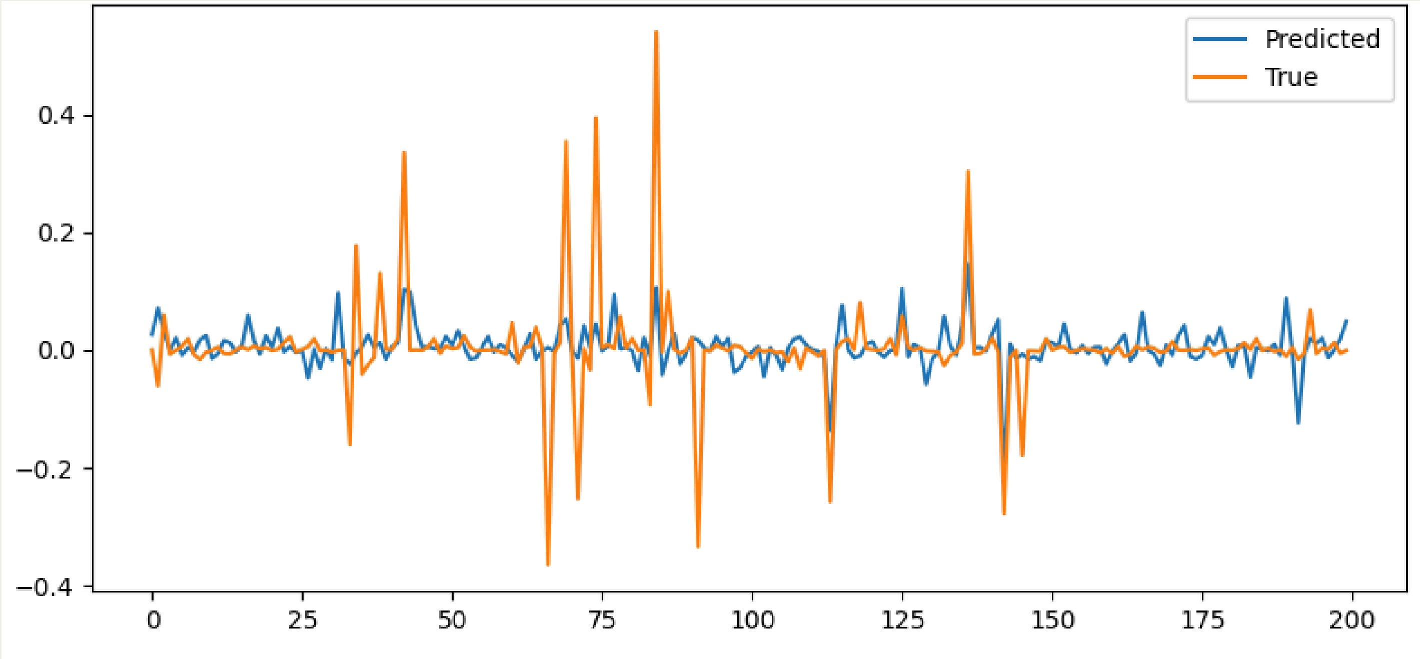
Our experiments and learning

Second set experiments –

Worked on learnings from the first set of experiments with different data fragments from different towns, weathers etc that was available.

Findings and Learnings:

- Dataset is skewed more towards straight line data and there is very less turn data available.
- Thus training on any random fragment of data will not help.
- Fragmented the data such that more learning is on turning data and some on straight line data



LABEL-EFFICIENT FINE-TUNING

Our experiments and learning

Final Set of Experiments

Worked on different fractions of turn data and straight data. Ended up with 50% of both such that car stays in line and turns well.

- Worked on experiments of choosing data equally from all the available data ranges such that car could perform well on turns, straight lines and the intermediary junctures between those.
- This experiment made the performance on turns worse.

Final Setting:

- 50% of each turn data and straight data in a dataset of ~10K images.
- The straight line images also include the intermediary junctures as well between the straight and turn as a part of them now.

The curse of this setting:

- Since, the intermediary juncture data was not specifically taken care of that much the car turns and then proceeds to hit the curb because that intermediary stretch was not generalised well.

Demo Links:

https://drive.google.com/file/d/1szZe35-DaQnRKZE5AUF8ADE9-T6qyRuL/view?usp=sharing

https://drive.google.com/file/d/1KQ2Q6DdwY2mlRaXluGG_H0DV0WUJZdzW/view?usp=sharing

DESIGN CHOICES THAT CHANGED THE GAME

Loss Tweaks:

- Switched from MAE to MSE to penalize large steering spikes more heavily.
- Experimented with combined MSE + cubic loss for sharper error handling.

Label Scaling:

- Scaled steering labels ×100 to improve numerical stability during training.

Split Model Strategy:

  - Identified dataset imbalance: turning events (spikes) were rare.
  - Created specialized models:
    - Only spikes (>7° steering)
    - Only straight lines (<7° steering)
    - Final hybrid: 50% spike + 50% straight merge for balanced specialization.

Other Trials:

  - Tested both MAE vs MSE loss functions.
  - Tried classification for turn detection + regression for fine steering prediction.
  - Explored multiple dataset splits (random frames, spikes-only, straight-only).

Thank you!