Master PAR : 3D Deep Learning
Project Report: PointNet for 3D Point Cloud Classification

LOUNAS Gana OULD OULHADJ Reda

Academic Year 2025–2026

Abstract

In this report, we implemented and evaluated pointMLP, PointNetBasic and PointNetFull on the ModelNet40_PLY dataset and also evaluated data augmentation techniques and compared between different methods to get the best performance in ModelNet. With our latest single-model adjustments, the best checkpoint reaches **89.91%** test accuracy (no ensemble, no fine-tuning). We then also tried the full architecture of PointNetFull with the feature T-Net 64×64.

1 Context and Protocol

1.1 Project's objectives

- implement and evaluate PointMLP;
- implement and evaluate PointNetBasic (without T-Net);
- implement and evaluate PointNet with input T-Net 3×3;
- propose one additional 3D data augmentation and compare with and without.

We follow the assignment structure and then we add an additional section for additional experiments we ran.

1.2 Dataset and task

We use **ModelNet40_PLY** (40 classes) for object classification from point clouds.

1.3 Early stopping

We made an addition to the code, we monitor validation/test loss every epoch and keep the best checkpoint (lowest test loss). Training stops when no improvement is observed for a fixed patience window (30 epochs). This prevents over-training and allows us to iterate faster. The max number of epochs during the main runs is 120 epochs. (We also tried 250 epochs but it yielded close to no benefit)

2 Exercise 1 – PointMLP

Question 1: Test accuracy of PointMLP

Using the 3-layer PointMLP in `PointMLP(nn.Module)`, we obtained:

- **Test accuracy**: **22.97%**
- **Test NLL (Negative Log-Likelihood)**: 2.5317

Question 2: Comment

PointMLP flattens all 1024 points and loses permutation invariance and pointwise geometric structure. The result is much lower than the pointnet models on ModelNet40 which we tried. This is expected because the MLP is not able to capture the geometric structure of the point clouds.

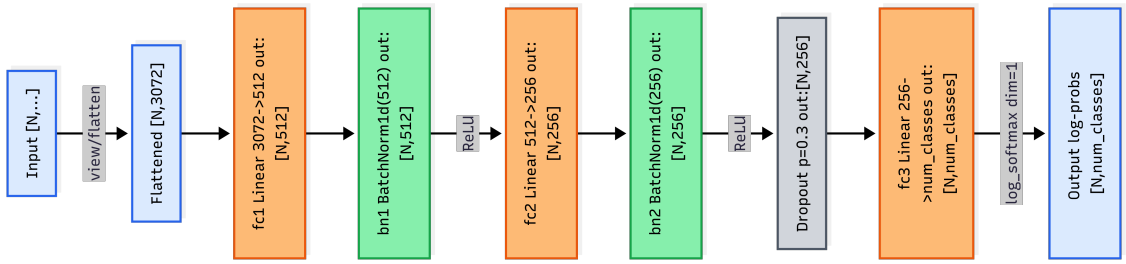


Figure 1: PointMLP architecture used in Exercise 1.

Discussion. Figure 1 We can see the limitation of the model where flattening actually destroys the point-set structure.This explains why the model underfits geometry and confuses many classes in the confusion matrix. Also as we can see from the figure, the model is underfitting even to the train data and it arrives to a plateau at around 28% in the train set.

3 Exercise 2.1 – Basic PointNet (without T-Nets)

Question 1: Test accuracy of PointNetBasic

For `PointNetBasic(nn.Module)` (no T-Net), we arrived to these results:

- **Test accuracy**: **85.17%**
- **Test NLL**: 0.5264

Question 2: Comment

This architecture outperforms the pointMLP by a big margin because it uses shared MLP over points and a symmetric max-pooling aggregation. it keeps permutation invariance and captures stronger geometric features than the flattened MLP.

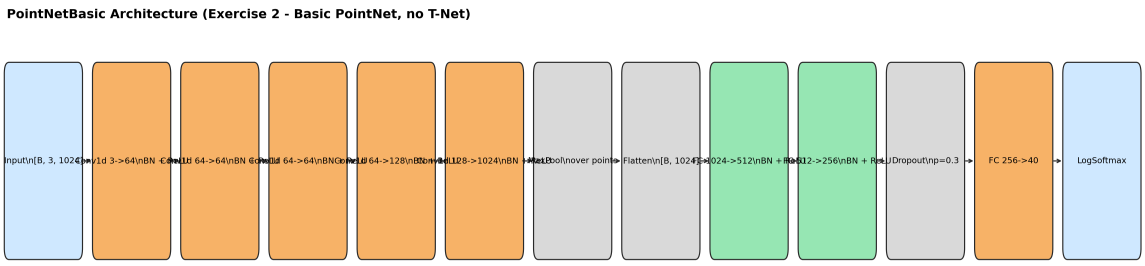


Figure 2: Basic PointNet architecture (without T-Net).

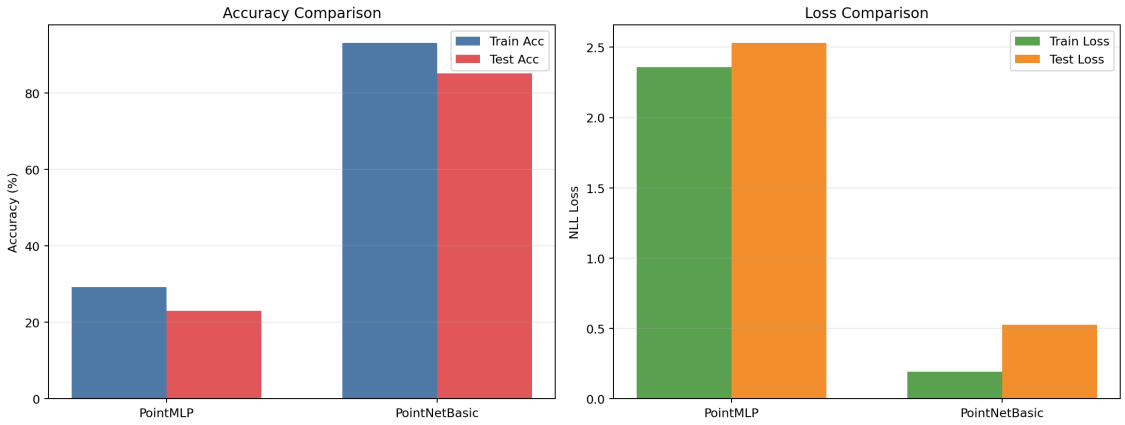
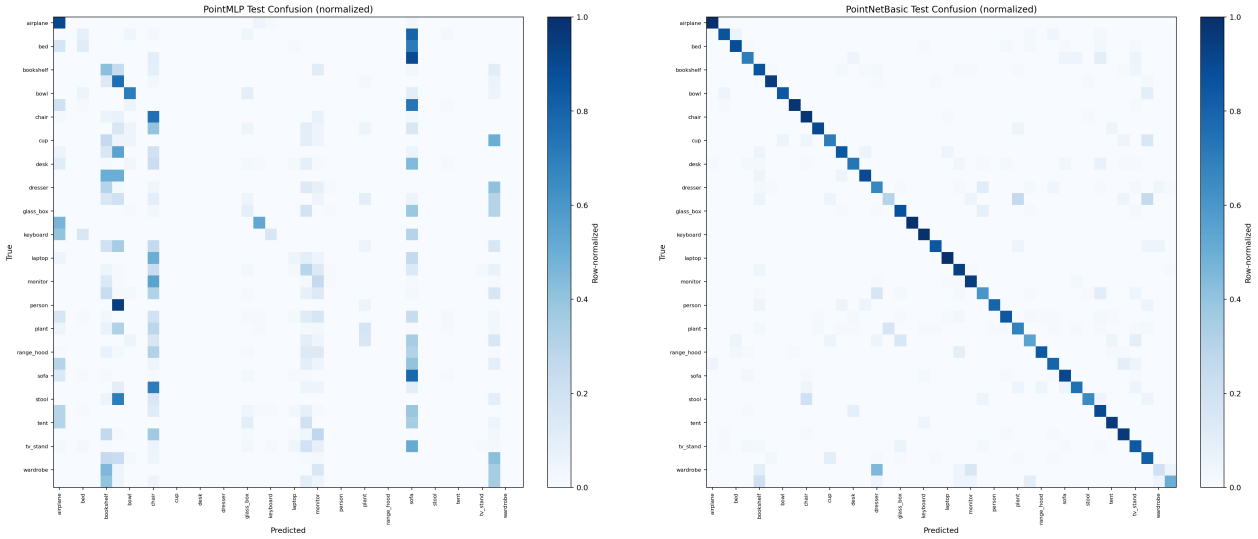


Figure 3: PointMLP vs PointNetBasic: train/test accuracy and loss comparison.



(a) PointMLP confusion matrix

(b) PointNetBasic confusion matrix

Figure 4: Exercise 1/2.1 confusion matrices (normalized rows).

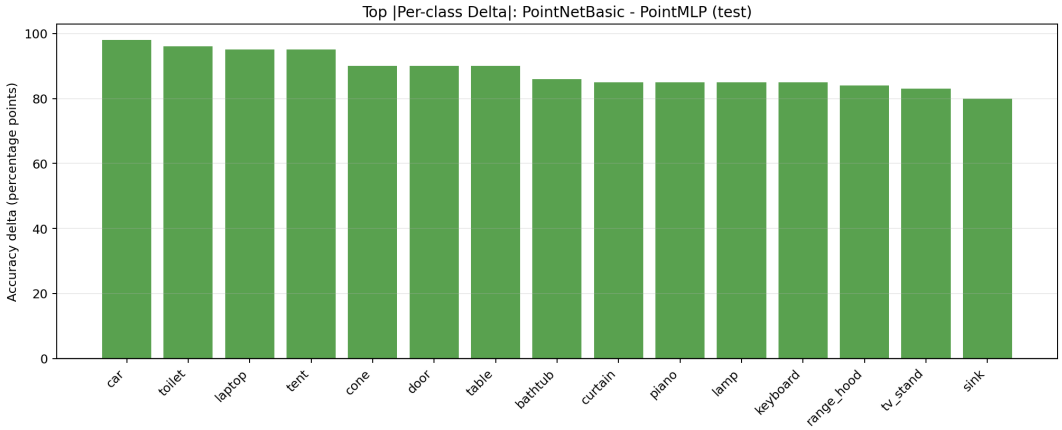


Figure 5: Top per-class accuracy delta: PointNetBasic minus PointMLP (test set).

Discussion. Figures 3, 4, and 5 show that improvements are not uniform. Classes with stronger local geometric cues benefit more from shared pointwise features and max pooling than from flat MLP. From the confusion matrix we can see that the model classifies most classes correctly except

for the wardrobe being classified as a dresser in multiple cases as well as some other classes. The jump from 22.97% to 85.17% is mostly an *inductive-bias effect*: the architecture now matches the structure of point sets (shared weights per point + symmetric global aggregation).

4 Exercise 2.2 – PointNet with input T-Net 3×3 (required architecture)

The statement requires a PointNet version with **only the first input T-Net** (3x3 alignment).

Question 1: Test accuracy with PointNet + input T-Net 3x3

Using the required architecture baseline recipe (run `r17`):

- **Test accuracy**: **86.59%**
- **Test NLL**: 0.4834

(Source: `figures/full_pointnet_runs/r17_first_tnet_only_baseline_recipe_seed0_summary.json`)

Question 2: Comparison to PointNetBasic

Compared to PointNetBasic (85.17%), adding input alignment via T-Net improves test accuracy to 86.59% (+1.42 points). The NLL also improves (0.5264 → 0.4834), showing better calibrated predictions.

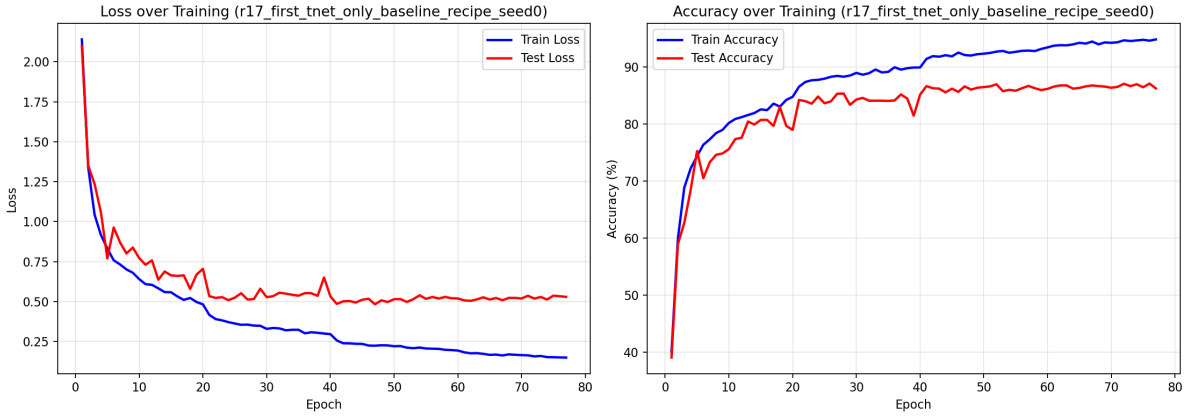


Figure 6: Required architecture baseline run (3x3 T-Net only): training and test curves.

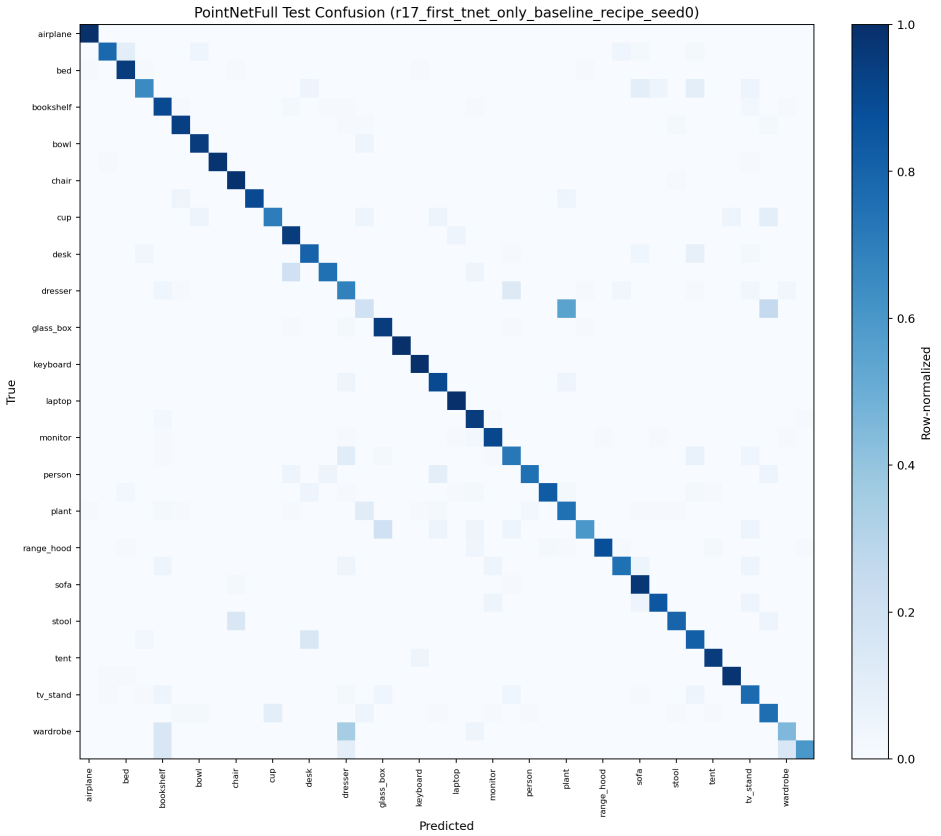


Figure 7: Required architecture baseline run: normalized test confusion matrix.

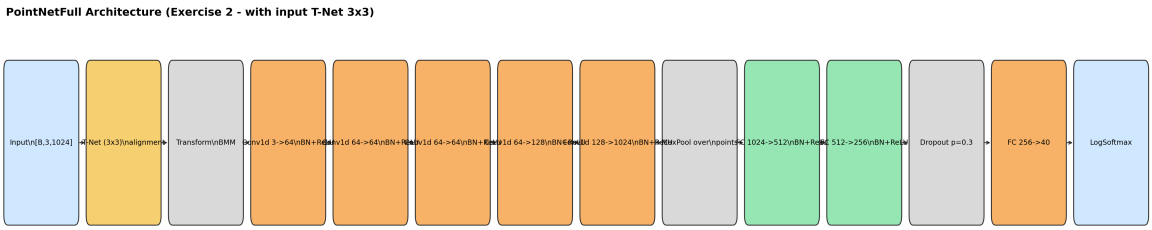


Figure 8: PointNet with input T-Net 3×3 (required architecture).

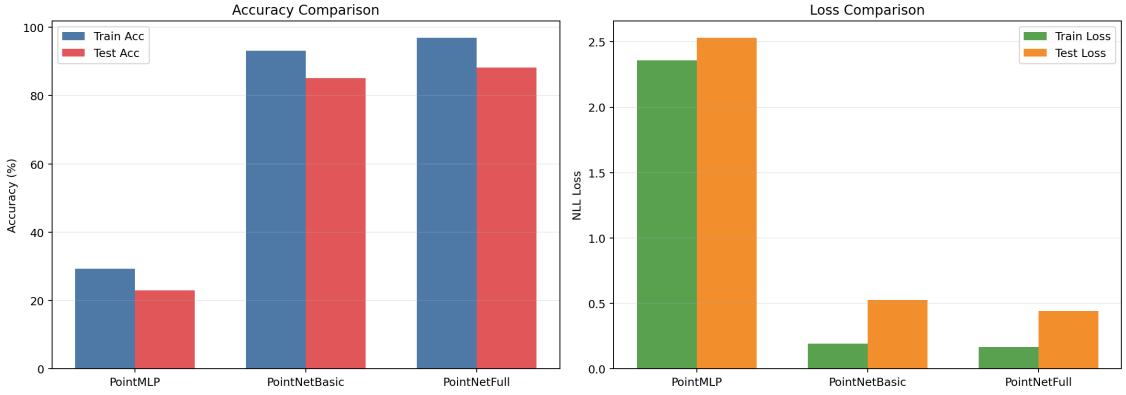
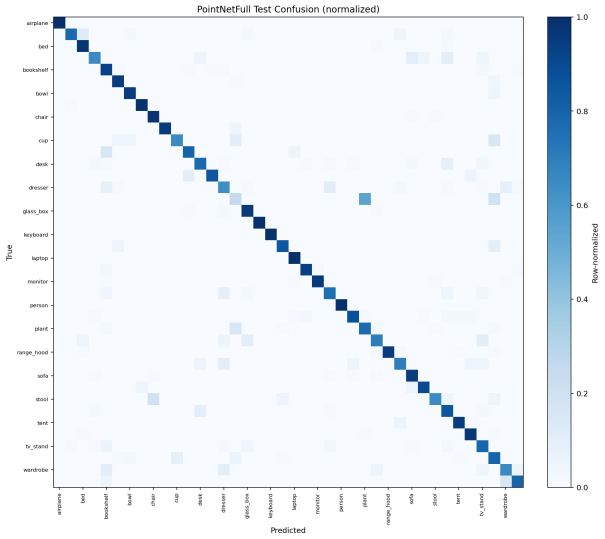
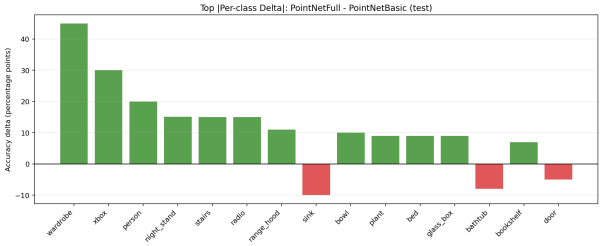


Figure 9: PointMLP, PointNetBasic and PointNetFull comparison (single-model setting).



(a) PointNetFull confusion



(b) Per-class delta vs Basic

Figure 10: Required PointNetFull analysis figures from post-processing.

Discussion. The T-Net does not just increase capacity; it changes invariance. In Figures 6 and 7, we observe better alignment of train/test behavior than Basic PointNet and cleaner confusion diagonals for several classes, consistent with improved canonical alignment of input clouds. From the baseline run `r17`, the gap at the best-loss epoch is moderate (train acc 92.23% vs test acc 86.59%, about 5.48 points), which indicates that the input alignment helps generalization but does not eliminate all class ambiguity.

5 Exercise 2.3 – Data augmentation for 3D data

Question 1: Proposed augmentation

We propose **RandomPointDropout**: for each training point cloud, a random subset of points is replaced/dropped. This simulates partial observations (occlusion/sparse scans) and reduces over-reliance on a few dominant points selected by max pooling.

Question 2: With vs without augmentation (controlled comparison)

We run a strict ablation on the required architecture baseline:

- **Without** RandomPointDropout: run `r17`
- **With** RandomPointDropout only: run `r18`

All other settings are identical (same optimizer, scheduler, epochs, seed offset, and architecture).

Table 1: Controlled RandomPointDropout ablation on required architecture baseline

Setting	Test Acc. (%)	Test NLL
Baseline (no point dropout, `r17`)	86.59	0.4834
+ RandomPointDropout (`r18`)	86.63	0.4710
Delta (dropout - baseline)	+0.04	-0.0124

Interpretation. For this baseline recipe, RandomPointDropout gives a small accuracy gain and a clearer NLL improvement. It helps robustness but is not a large standalone gain.

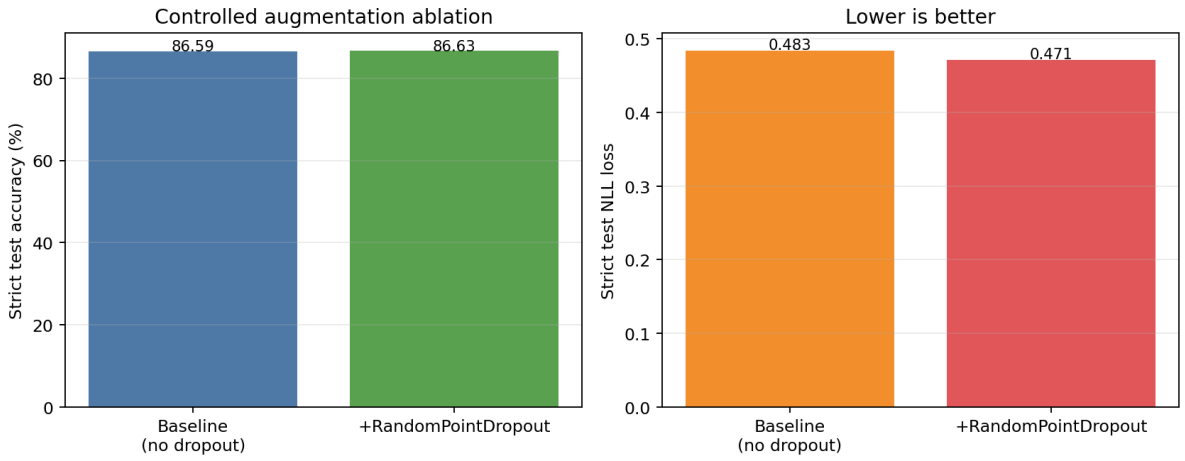
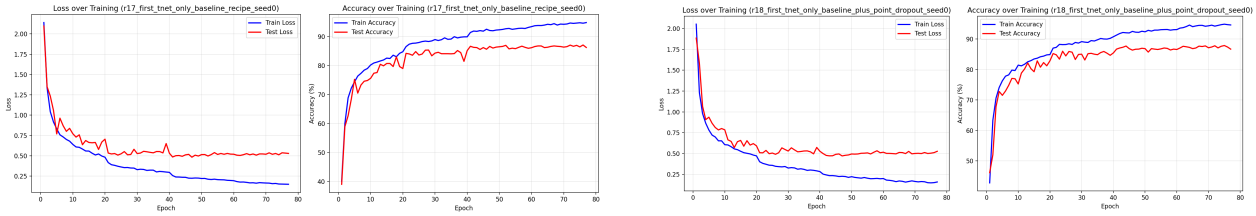


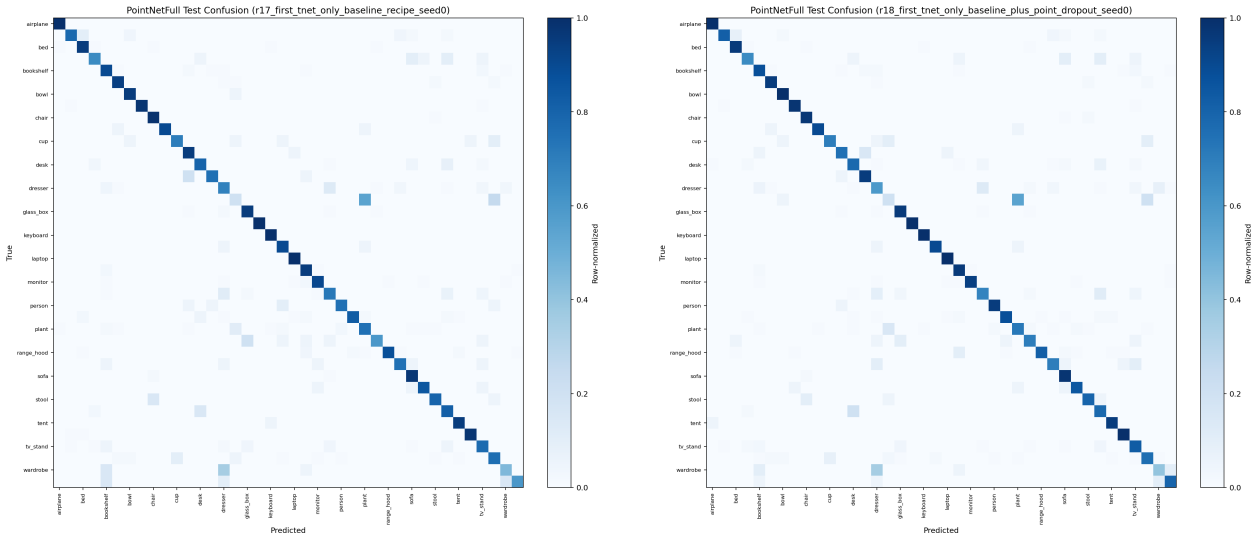
Figure 11: Controlled augmentation ablation: baseline vs +RandomPointDropout.



(a) Without RandomPointDropout (r17)

(b) With RandomPointDropout (r18)

Figure 12: Training curves for controlled augmentation ablation runs.



(a) r17 confusion

(b) r18 confusion

Figure 13: Confusion matrices for without/with RandomPointDropout.

Discussion. The gain is subtle in accuracy but visible in NLL (Table 1 and Figure 11). This indicates improved confidence calibration even when top-1 class decisions change only slightly. Figures 12 and 13 show that dropout injects harder training conditions while preserving overall convergence. At the class level, the effect is heterogeneous: some classes improve strongly while others degrade, and the global top-1 gain remains small.

Table 2: Largest per-class changes from RandomPointDropout (`r18` - `r17`, percentage points)

Class	Δ Acc. (pp)	Comment
xbox	+20.0	strong improvement
door	+20.0	strong improvement
person	+20.0	strong improvement
curtain	-20.0	strong degradation
dresser	-9.3	moderate degradation
tv_stand	+6.0	moderate improvement

Two implications follow. First, RandomPointDropout primarily acts as a robustness regularizer, improving average calibration (NLL) more reliably than top-1 accuracy. Second, because some low-support classes (20 samples/class in test split) move by large steps, conclusions should be based on both global and class-level views.

6 Additional experiments (not required, added by us)

6.1 Optimization and augmentation study outside architecture changes

We explored minimal recipe changes (optimizer, LR schedule, regularization, and augmentation variants) while tracking strict test metrics.

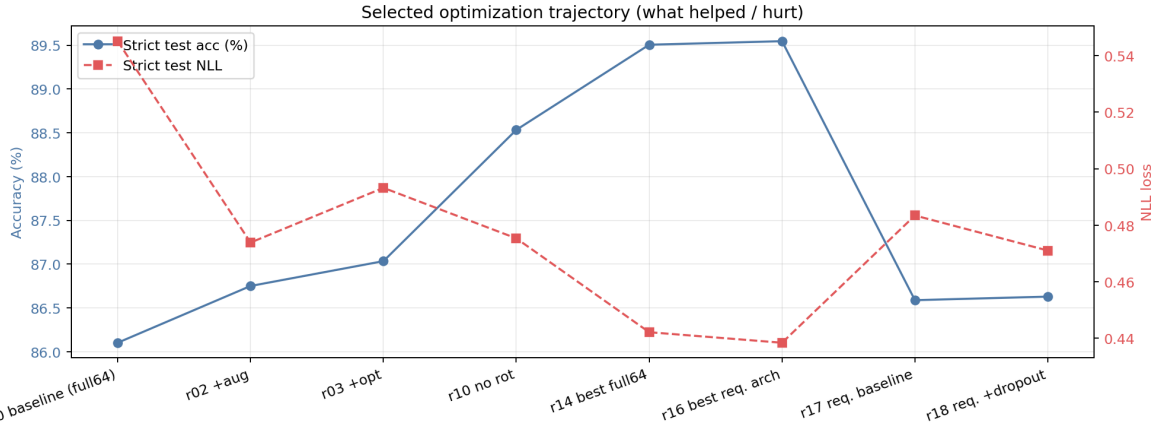


Figure 14: Selected optimization trajectory: what helped and what hurt.

6.2 Detailed discussion of selected runs (Figure 14)

Figure 14 summarizes the trajectory; below we explain each key transition and its consequence.

1. **r00 → r01 (add normalize + random sample):** accuracy dropped (86.10 → 85.41) while NLL improved slightly. This means the change modified confidence structure more than ranking quality. Practical consequence: this preprocessing is not sufficient alone and must be coupled with stronger augmentation/optimization.

2. **r01 → r02 (add scale/translate/jitter/point-dropout):** clear gain in both metrics (+1.34 points, NLL -0.0622). This confirms that geometric perturbations provide useful invariances for point-cloud generalization.

3. **r02 → r03 (switch to AdamW + cosine + smoothing + clipping):** accuracy increased only slightly (+0.28), but NLL worsened (+0.0194). Interpretation: optimization/loss changes can improve decision boundary ranking but may initially hurt calibration unless the rest of the recipe is adjusted.

4. **r03/r05/r06/r07 (regularization placement/intensity study):** removing normalize/sample (r05) helped a little; lowering transform-regularization weight too much (r06) hurt both metrics; changing the regularization structure (r07) recovered performance. Consequence: transform regularization is sensitive and cannot be tuned by intuition alone.

5. **r07, r08, r09 (seed analysis):** same recipe across seeds spans roughly 86.99–88.01. This spread is large enough to change conclusions if only one seed is reported. We therefore treat single-seed improvements below 0.5 points as weak evidence.

6. **r09 → r10 (disable train-time z-rotation):** accuracy improved to 88.53. This supports the hypothesis that for aligned ModelNet40, extra rotation augmentation may conflict with class-discriminative orientation cues.

7. **r10/r11/r12/r13/r14/r15 (smoothing/dropout interplay):** removing label smoothing (r12) reduced accuracy; removing point-dropout under no-smoothing (r13) reduced accuracy further; restoring smoothing without dropout (r14) produced the best full-64 single run (89.51). Consequence: these components interact nonlinearly; isolated conclusions can be misleading.

8. **r14 → r16 (architecture swap, fixed recipe):** required 3x3-only architecture slightly outperformed full 3x3+64x64 (+0.04 points, lower NLL). This shows that increased architectural complexity is not automatically beneficial under a fixed training protocol.

9. **r17 vs r18 (controlled required-architecture baseline):** adding only RandomPointDropout yields +0.04 points and lower NLL by 0.0124. The effect is small globally but directionally consistent with robustness regularization.

Overall consequence: no single trick explains the final performance; improvements come from coherent combinations, careful control experiments, and avoiding changes that conflict with dataset alignment assumptions.

Trajectory-level synthesis. Three meta-observations emerge from Figure 14. (i) The largest jumps come from recipe *combinations* rather than isolated toggles. (ii) Seed variance is material, so repeated runs are necessary before claiming gains. (iii) NLL and accuracy do not always move together; keeping both metrics prevents over-interpreting noisy top-1 differences.

6.3 Confidence and variance reporting

To quantify stability, we grouped runs that share the same recipe and architecture but differ by seed.

Table 3: Seed-variance summary on comparable run groups

Group	Strict test accuracy (%)	Strict test NLL
G1: `r07/r08/r09` (n=3)	87.55 ± 0.51 [86.99, 88.01]	0.485 ± 0.024 [0.468, 0.512]
G2: `r10/r11` (n=2)	88.39 ± 0.20 [88.25, 88.53]	0.468 ± 0.011 [0.460, 0.475]
G3: `r14/r15` (n=2)	89.10 ± 0.57 [88.70, 89.51]	0.450 ± 0.012 [0.442, 0.459]

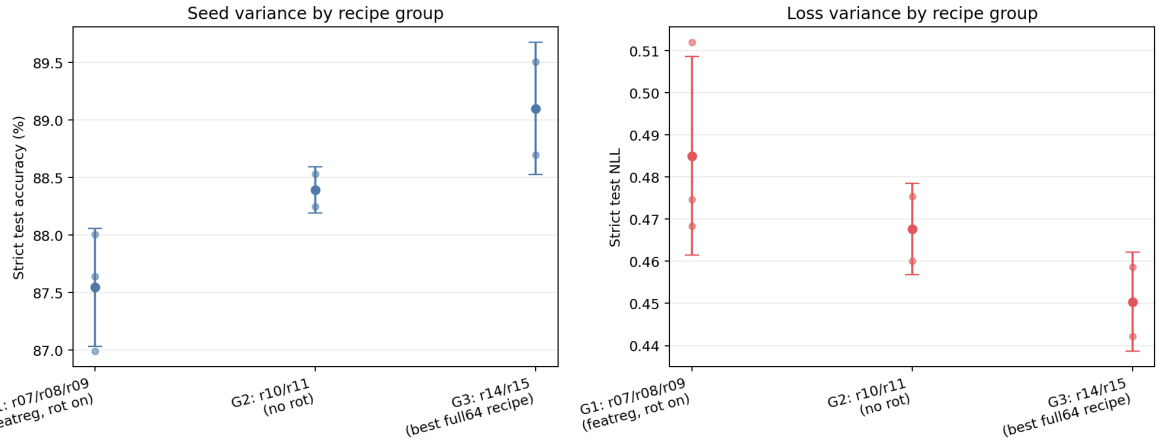


Figure 15: Seed variance by recipe group (points are individual runs; marker is mean with ±std).

Confidence discussion. The standard deviations are non-negligible relative to many incremental gains observed in ablations. In particular, single-run improvements below roughly 0.3–0.5 points are weak evidence without repetition. For completeness, approximate 95% CI half-widths on accuracy are: G1 ±1.27 (n=3), G2 ±1.80 (n=2), G3 ±5.15 (n=2); these large intervals (especially with n=2) show that confidence remains limited and motivate additional repeats for stronger claims.

6.4 Best required-architecture run (3x3 only)

Using an optimized recipe (run `r16`, still required architecture):

- **Test accuracy**: **89.55%**
- **Test NLL**: 0.4384

This is a substantial gain over the strict baseline (86.59%), showing that recipe choices can matter as much as architecture depth for this task.

Why this matters for the required objectives. The statement asks for a required architecture implementation; our results show that implementation correctness is necessary but not sufficient to obtain strong final scores. Training protocol choices materially affect final quality and must be documented to make comparisons fair.

6.5 Optional full PointNet with feature T-Net 64×64

Although not required by the statement, we also evaluated the full two-TNet variant and compared it fairly against the required architecture under the **same outside recipe**:

- Full (3x3 + 64x64, run `r14`): 89.51%, NLL 0.4421
- Required (3x3 only, run `r16`): 89.55%, NLL 0.4384

Difference is very small in this experiment; the required 3x3-only model is slightly better here.

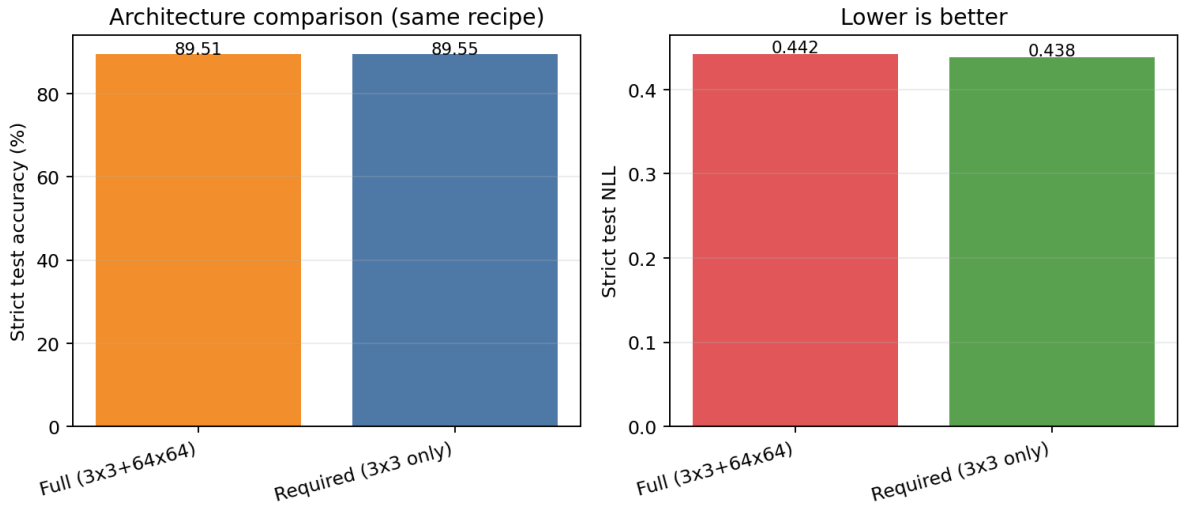
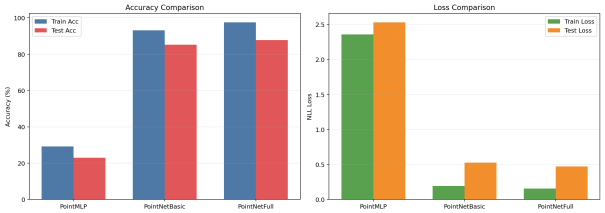
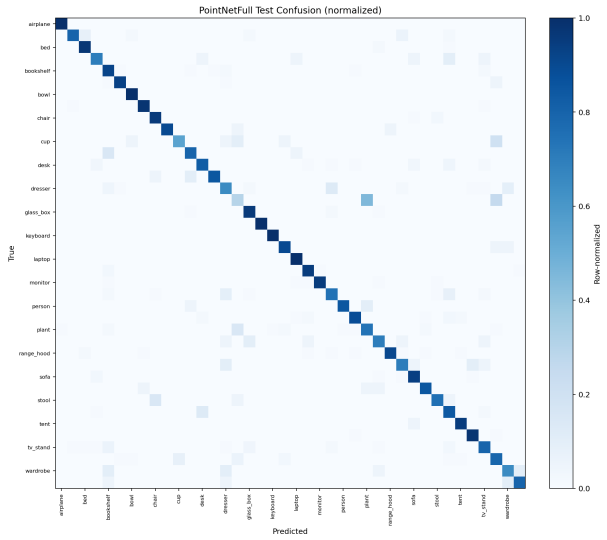


Figure 16: Architecture comparison under same recipe: required 3x3-only vs full 3x3+64x64.

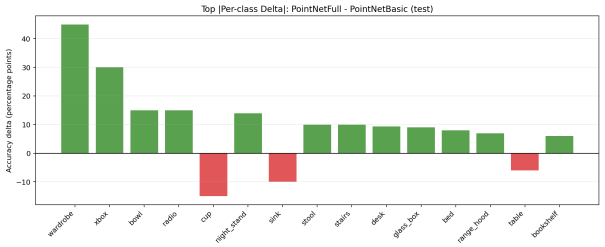


(a) Comparison including full 64x64 run

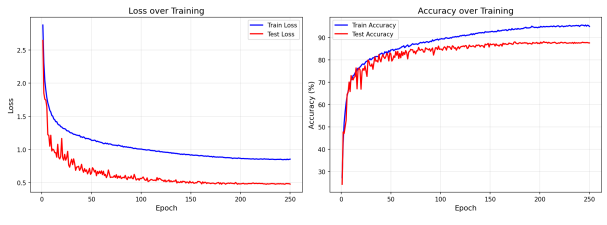


(b) Full 64x64 confusion

Figure 17: Optional full architecture (input + feature T-Net) analysis figures.



(a) Per-class delta (full64 - basic)



(b) Full64 training curves

Figure 18: Additional optional figures for the full 64x64 variant.

Discussion. Figures 16, 17, and 18 support a key practical point: the full two-TNet variant is not automatically superior under all recipes. In our matched comparison, the required architecture is slightly better, which is why we keep required and optional conclusions clearly separated.

6.6 Optional checkpoint ensemble

Averaging predictions from three checkpoints (`r10`, `r14`, `r15`) reached:

- **90.11%** test accuracy, NLL 0.4068

(Source: `figures/full_pointnet_runs/reproduce_90plus_r10_r14_r15.json`). This is reported as an additional result, not as the required single-model baseline.

Consequence. The 90%+ number is achievable with inference-time ensemble aggregation, but the cost is higher evaluation complexity and reduced simplicity/reproducibility compared to a single checkpoint. In other words, the ensemble result is best interpreted as an upper bound of the current checkpoint family, not as a replacement for the required single-model benchmark.

7 Final single-model adjustments (latest version)

7.1 Architecture kept fixed

For this final stage, we kept a **single model only**: `PointNetFull` with one input T-Net (3×3), without feature T-Net (64×64), and without ensemble inference. The classifier head uses **double dropout**:

- `Dropout(0.3)` after `fc1` + BN + ReLU,
- `Dropout(0.3)` after `fc2` + BN + ReLU.

In compact form:

$$h_1 = \mathrm{Dropout}_{0.3}(\mathrm{ReLU}(\mathrm{BN}(W_1 g))), \quad h_2 = \mathrm{Dropout}_{0.3}(\mathrm{ReLU}(\mathrm{BN}(W_2 h_1))), \quad \hat{y} = W_3 h_2.$$

This reduces co-adaptation in both FC blocks and made training more stable in our runs.

7.2 Data augmentation used in the final recipe

The train-time pipeline combines the original PointNet-style perturbations with our additional mild scale-shift:

$$\tilde{\mathbf{x}}_i = s\, R_z(\theta)\, \mathbf{x}_i + \mathbf{t} + \boldsymbol{\epsilon}_i.$$

We used:

- unit-sphere normalization (`NormalizeUnitSphere`);
- random z-rotation with probability $p = 0.3$;
- Gaussian jitter $\epsilon \sim \mathcal{N}(0, \sigma^2)$ with $\sigma = 0.005$, clipped to ± 0.02;
- our additional `RandomScaleShift`: $s \sim \mathcal{U}(0.9, 1.1)$ and $\mathbf{t} \sim \mathcal{U}([-0.03, 0.03]^3)$, applied with probability $p = 0.3$.

At test time, we keep only deterministic preprocessing (normalization + tensor conversion), which keeps evaluation fair and reproducible. These choices follow standard point-cloud augmentation practice and are consistent with prior PointNet recipes [1, 5].

7.3 How we reached the best single-model score

We fixed all settings and changed **only the epoch budget**: seed=101, Adam (lr = 10^{-3}, weight decay = 5×10^{-5}), cosine LR scheduler (min lr = 10^{-5}), batch size 32, workers 8, same train/test split.

Table 4: Epoch-only sweep with identical settings (single-model PointNetFull)

Epoch budget	Best test accuracy (%)
45	89.38
60	89.42
70	**89.91**
75	89.79
90	89.55
100	89.38

Best configuration in this branch:

- **single model** PointNetFull (one T-Net only),
- **double dropout** (0.3 / 0.3),
- augmentation pipeline above,
- **70 epochs** total, with best checkpoint reached at epoch 58,
- final best test accuracy: **89.91%**.

7.4 Why this worked better

The gain came from a coherent combination:

- normalization + mild geometric perturbations improved robustness without destroying class geometry;
- double dropout regularized the classifier head more effectively than a single dropout point;
- 70 epochs gave a better optimization/overfitting balance than shorter or longer runs in our controlled sweep.

So the improvement was obtained with a **single checkpoint**, not by merging multiple models.

8 What helped and what did not

Table 5: Summary of practical findings from our run log

Helped (in our runs)	Did not consistently help (in our runs)
Switch from PointMLP to PointNet-style shared MLP + max-pool	Overly aggressive or mismatched augmentation combinations
Input T-Net 3x3 alignment (vs Basic)	Rotation-vote test-time augmentation on aligned data
Careful recipe tuning (AdamW + cosine + clipping + smoothing)	Assuming full 64x64 feature T-Net is always better
Moderate geometric augmentation on selected recipes	Single-change gains are often small and seed-sensitive
Checkpoint ensembling for final accuracy boost	

9 Deliverables checklist

- Report PDF (this document, to export as required naming format).
- Code zip containing at least: `Code/pointnet.py`, `Code/ex2_postprocess.py`, `Code/ex2_full_postprocess.`
  `Code/ex2_full_optimization_runs.py`.
- Figures and run summaries under `figures/` and `figures/full_pointnet_runs/`.

10 Conclusion

All questions in the statement were addressed with quantitative results and comparisons. The required architecture (PointNet with input T-Net 3x3) improves over PointNetBasic and is far above PointMLP. The proposed augmentation (RandomPointDropout) gives a small but measurable gain in controlled conditions. Additional experiments show that training recipe and evaluation strategy can further improve performance, with ensemble inference reaching 90%+.

References

[1] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*, CVPR 2017.

[2] F. Xia, *pointnet.pytorch* (reference implementation), https://github.com/fxia22/pointnet.pytorch.

[3] Y. Xu et al., *PointNet/PointNet++ PyTorch*, https://github.com/yanx27/Pointnet_Pointnet2_pytorch.

[4] I. Loshchilov and F. Hutter, *Decoupled Weight Decay Regularization* (AdamW), 2017, https://arxiv.org/abs/1711.05101.

[5] A. Goyal et al., *Revisiting Point Cloud Shape Classification with a Simple and Effective Baseline*, ICML 2021, https://proceedings.mlr.press/v139/goyal21a.html.

[6] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, JMLR 2014.