Mini-Project Brief:
Self-Supervised Learning on Provided Datasets

Abstract

This document specifies the eight-week mini-project focused on applying three self-supervised learning (SSL) methods to a provided dataset (or dataset pair). It defines task objectives, deliverables, evaluation met-rics, checkpointing/ resumability requirements, marking scheme, and reporting stan-dards.

1. Goal and Setting

Over eight weeks, teams will execute a complete study applying three SSL meth-ods to a provided dataset (or dataset pair). The work must be fully reproducible on Kaggle; Colab or local experimentation is optional for comparison only. Submit Kaggle notebooks, a paper-style report, slides, and saved model artifacts.

2. Task 1 — EDA and Related Work

2.1. Image-focused EDA (with concise figures/tables)

Report: (i) RGB and HSV histograms; per-image and per-class mean/standard deviation; brightness/contrast spread; saturation clipping; (ii) resolution and aspect-ratio distributions; resizing/padding strategy and class-wise effects; (iii) sharpness/noise using Laplacian variance and a light noise proxy; (iv) white-balance sanity via a gray-world check; (v) duplicate detection using perceptual hashing; (vi) class balance; (vii) leakage safeguards in splitting (e.g., grouping by source/ID). Run a small augmentation probe (crop/flip/color jitter/blur) to identify safe vs. harmful augmentations.

2.2. Related Work (dataset/domain)

Create a *Related Work Summary* focused on the same dataset or a closely related domain. This table *constrains Task 2*: supervised backbones must be drawn from credible related work on this dataset/domain.

Required Table Columns (exact). *Title — Dataset name and URL — Dataset description (samples, classes, images per class or per split) — Methods name — Accuracy of the model — Pros — Cons — Citation.*

A LaTeX template is provided in Section 10.

Deliverables: one Kaggle EDA notebook; 1–2 page related-work summary with the required table and citations.

3. Task 2 — Supervised Baselines from Related Work

Implement at least five pretrained supervised backbones *explicitly* drawn from models reported in the Task-1 Related Work table (e.g., GoogLeNet, MobileNet, ResNet-50/101, EfficientNet, ViT—only if the related work in this domain used them).

Splits: evaluate across {90:10, 80:20, 70:30, 60:40, 50:50, 40:60, 30:70, 20:80, 10:90} (test set fixed per ratio). Within each training portion, allocate 10% for validation.

Training: train until a clear learning-curve plateau or for at least 50 epochs; show training/validation curves.

Report (per model and split): Accuracy, per-class Accuracy, Precision, Recall, F1-score, ROC–AUC with ROC curves, Confusion Matrix, training/testing wall-clock time, and GFLOPs per inference at a fixed input size (state the size). Summarize failure modes (confused class pairs). Identify the best supervised baseline as the reference for SSL.

Deliverables: Kaggle notebook(s) with consistent logs; a brief table highlighting the top-1 supervised result.

4. Task 3 — Select and Explain Three SSL Methods

Choose three SSL methods suitable for your data (e.g., SimCLR, BYOL, Barlow Twins, DINO/DINOv2, iBOT, MAE, VICReg, SimSiam, SwaV). Justify each selection based on dataset characteristics. Prepare a teaching-style explanation (as if to a class-10 student): intuition, objective (with a friendly diagram), how positives/negatives or masking/tokenization work, and why the method fits the dataset.

Deliverables: 6–8 explanatory slides (or a narrative section) with citations.

5. Tasks 4–6 — Implement SSL-1, SSL-2, SSL-3

5.1. Label-free Pretraining

Run the SSL pretext on the training split with a documented augmentation recipe. Save the frozen encoder.

5.2. Downstream Evaluation

- **Linear probe** on frozen features (primary SSL yardstick).
- **Shallow heads:** MLP, SVM, Decision Tree, Random Forest on frozen features.
- **Full fine-tuning** of the encoder; compare to linear probe to quantify gains/costs.

5.3. Embedding Analysis

Provide t-SNE, UMAP, and PCA plots colored by true labels; report the Silhouette score to quantify separation.

5.4. Metrics & Logs

Accuracy, per-class Accuracy, Precision, Recall, F1, ROC–AUC with curves, Confusion Matrix, learning curves (pretraining proxy + downstream), train/test time, and GFLOPs (same input size across methods). Add k-NN accuracy ($k \in \{1, 5, 20\}$) in embedding space and label-efficiency curves (linear-probe accuracy at $\{1\,\%, 5\%, 10\,\%, 25\,\%, 50\,\%\}$ labeled data).
Deliverables: one Kaggle notebook per SSL method; a brief comparison table.

6. Task 7 — Ratios, Ablations, and Statistics

Re-run downstream evaluations for all SSLs across the same train:test grid used in Task 2. Perform at least one solid ablation per SSL (e.g., remove a key augmentation; change projection-head depth; vary batch size or pretraining epochs; alter optimizer/LR). Sum-marize effects on linear-probe and label-efficiency outcomes.

Statistics: use paired t-tests for two-model comparisons on identical folds; for three or more models (or multiple datasets), use Wilcoxon/Friedman with Nemenyi post-hoc when appropriate. Report p-values and discuss practical vs. statistical significance.
Deliverables: a consolidated results notebook with ablation tables, statistical-test outputs, and 1–2 figures showing effect sizes.

7. Task 8 — Paper-Style Report and Final Presentation

Prepare an 8–12 page report: abstract, introduction, related work, methods (supervised baselines + three SSL methods), experimental setup, results (including embeddings and label-efficiency), ablations with statistics, discussion, limitations/ethics, and conclusion. Create 10–12 slides for a concise, defensible presentation.
Deliverables: PDF report, slides (PDF/PPTX), public Kaggle project with "Run All" notebooks, and a mirroring GitHub repo (requirements, fixed seeds, README).

8. Model Saving, Checkpointing, and Resumability (Mandatory)

8.1. Where to Save

Save checkpoints to `/kaggle/working/` during training. At notebook end, persist artifacts by attaching them as notebook Outputs (so they remain available on rerun and can be converted into a Kaggle Dataset).

8.2. What to Save (Supervised and SSL)

Save both the **latest** and the **best** checkpoint (based on your validation metric). Each checkpoint must include:

- `model_state_dict` (include encoder and any projection/head modules),
- `optimizer_state_dict`, `scheduler_state_dict` (if used),
- `epoch`, `global_step`, `best_metric` (name and value),
- random states (Python, NumPy, PyTorch CPU/GPU),
- data split manifest (file list or seed + stratification info),
- method-specific buffers (e.g., EMA/momentum teacher for BYOL/DINO; queues/memory banks for MoCo/SwaV; tokenizers/positional embeddings for ViTs).

8.3. Best-Model Criterion

Supervised: best validation ROC–AUC (or top-1 accuracy when class imbalance is negligible—state and justify).
SSL: best *linear-probe* validation accuracy (use a consistent labeling budget across methods).

8.4. Resumable Training Loop (SSL required; supervised recommended)

Training must support *resume-from-checkpoint* restoring model, optimizer, scheduler, EMA/momentum teacher (if any), queues/buffers, epoch/step, RNG states, and logging offsets. Implement periodic and on-improvement saves; use early stopping with patience and save the best checkpoint. On resume, continue seamlessly and keep updating "latest" and "best".

8.5. Determinism and Logging

Fix seeds; enable deterministic flags when compatible. Record library versions and CUDA/cuDNN info. Log metrics and hyperparameters to CSV/JSONL and save alongside checkpoints.