Machine Learning Engineer Assessment

Activation Swap + Fine-Tuning Challenge

Time: 60–90 minutes
Format: Take-home Jupyter notebook
Submission: Email **one executable `.ipynb`** (with code, outputs, and brief analysis) to **rocio.novo@skylarklabs.ai** by **September 10, 2025**.
File name: `ML_Assessment_<YourName>.ipynb`

Environment & Runtime Expectations

- Python 3.9+; PyTorch 2.x; torchvision 0.15+.
- The notebook **must run top-to-bottom without manual edits** on CPU (GPU optional).
- The notebook should **auto-download CIFAR-10** via `torchvision.datasets` if not present.
- Total runtime target: **≤ 20 minutes on CPU** (use small/efficient settings).

Notebook Structure (Required Cell Blocks) & Expected Outputs

0) Setup & Reproducibility (before Task 1)

What to implement

- Imports, device selection, and a single place to set seeds:
  - `random`, `numpy`, `torch`, `torch.backends.cudnn`, `torch.manual_seed`, etc.
  - Enable deterministic behavior where feasible.

Expected printed outputs

- **Environment summary** (Python, torch, torchvision versions; device).
- **Fixed seeds confirmation** (print the chosen seed).
- **Determinism note** (whether `cudnn.deterministic=True` and `benchmark=False`).

---

1) Model Architecture

1.1 Base Model Selection

What to implement

- Load one of: `resnet18`, `resnet34`, or `resnet50` from `torchvision.models`.
- Optionally load pretrained weights (justify briefly in a markdown cell).

Expected printed outputs

- **Model choice & pretrained flag** (e.g., "Using resnet18, pretrained=True").
- **Parameter count** and **trainable parameter count**.

1.2 Activation Function Replacement

What to implement

- **Depth-first traversal** over model modules.
- **Count all `nn.ReLU` occurrences** deterministically.
- **Replace ReLU from the 7th occurrence onward** with an alternative activation (e.g., `nn.SiLU()`, `nn.GELU()`, or `nn.LeakyReLU(0.01)`), **keeping the first 6 intact**.
- Replacement must be **module-level** (no functional API hacks).

Expected printed outputs

- **Total `nn.ReLU` count before replacement** (integer).
- **Indices of replaced activations** (e.g., a list starting at index 6 in 0-based or 7 in 1-based—state which you use).

- **Sanity check after replacement:** recount and report
    - “ReLU remaining: X”
    - “<AltActivation> inserted: Y”
- **Assertion** that exactly the intended number were replaced.

1.3 Classification Head Modification

What to implement

- Replace the final fully connected (FC) layer to output **2 or 3 classes**.
- Ensure **input features** → **output features** are dimensionally valid.

Expected printed outputs

- **Old head shape** (e.g., `in_features=512 → out_features=1000`).
- **New head shape** (e.g., `in_features=512 → out_features=3`).
- **Number of target classes**.

---

2) Data Pipeline

2.1 Dataset Preparation

What to implement

- Load **CIFAR-10** (train/test).
- Choose **2–3 classes** (e.g., `["cat", "dog", "airplane"]`).
- **Filter** the dataset to only these classes.
- **Relabel** them to contiguous IDs in `{0, 1, …, N_CLASSES-1}`.
- Split into **train/validation** (e.g., 90/10 or 80/20) or use CIFAR train as train and CIFAR test as val; state which.

Expected printed outputs

- **Selected classes** (original names).
- **Mapping table**: `{original_label_id: new_label_id}` and `{class_name: new_id}`.
- **Counts per class** for train and validation (e.g., a small table).

2.2 Data Augmentation

What to implement

- Reasonable transforms:
    - **Train**: random crop/resize, random horizontal flip, normalization.
    - **Val**: center/resize (no augmentation), normalization.

Expected printed outputs

- **Transform summary** for train and val (human-readable).
- **Sample batch shape** printed from each DataLoader.
- *(Optional but nice)*: A small **grid of 8–16 augmented train images** with their **new labels**.

---

3) Training & Evaluation

3.1 Training Loop

What to implement

- Standard supervised loop with:
    - Cross-entropy loss
    - Optimizer (e.g., SGD or Adam) + learning rate
    - Optional LR scheduler
- Track **epoch-wise**: train loss, train accuracy, val loss, val accuracy.

Expected printed outputs
For each epoch, a single line like:

```
Epoch 1/NN | Train Loss: 1.234 | Train Acc: 67.8% | Val Loss: 0.987
| Val Acc: 72.4%
```

- At the end, print **best validation accuracy** and the **epoch it occurred**.

3.2 Performance Reporting

What to implement

- Final validation metrics (accuracy mandatory; F1 optional if helpful).
- Brief markdown analysis (2–6 bullets) commenting on:
  - Effect (hypothesized) of activation swap
  - Any over/underfitting signs
  - Next steps you'd try with more time

Expected printed/displayed outputs

- **Final Val Accuracy** (e.g., "Final Val Acc: 78.3%").
- **Confusion matrix** (small plot or printed array) for 2–3 classes.
- *(Optional bonus)*: **Classification report** (precision/recall/F1).

---

4) Code Quality & Reproducibility

What to implement

- Clean structure: helper functions for
  - model traversal & replacement,
  - dataset filtering & relabeling,
  - training & evaluation.

- Error handling (e.g., assert class names exist, assert counts nonzero).
- One top-level `SEED = <int>` variable used everywhere.

Expected printed outputs

- A **final "Reproducibility summary"** block echoing:
  - Seed value
  - Any sources of nondeterminism you could not eliminate (brief note)
- **Total runtime** (print elapsed minutes/seconds).

What We Evaluate

- **PyTorch model manipulation:** correct, module-level activation swaps from the 7th ReLU onward.
- **Data pipeline:** correct filtering, relabeling, and reasonable augmentations.
- **Training correctness:** stable loop, meaningful metrics, and sanity checks.
- **Experimental clarity:** concise observations tied to results.
- **Code quality:** readability, structure, and reproducibility.

Quick Submission Checklist

- Single `.ipynb` with **all cells executed** and outputs visible.
- Runs **top-to-bottom** on CPU without edits.
- Printed **ReLU counts before/after** and **indices replaced**.
- Printed **head shapes (old → new)** and **class mapping table**.
- Printed **per-epoch metrics** and **final val accuracy**.

- Included **confusion matrix** and brief **analysis bullets**.
- Included **seed & environment summary** and **runtime**.

Example Output Snippets (format guide only)

- `Total nn.ReLU before replacement: 20`
- `Replacing activations at indices (1-based): [7, 8, 9, …, 20] with nn.SiLU`
- `Head: 512 → 3`
- `Selected classes: ['cat', 'dog', 'airplane']`
- `Mapping: {'airplane':0, 'cat':1, 'dog':2}`
- `Train counts: {0: 4500, 1: 4600, 2: 4400} | Val counts: {0: 500, 1: 500, 2: 500}`
- `Epoch 5/10 | Train Loss: 0.61 | Train Acc: 81.2% | Val Loss: 0.58 | Val Acc: 80.4%`
- `Best Val Acc: 82.1% (epoch 9)`
- `Seed: 1337 | cudnn.deterministic=True | cudnn.benchmark=False`