Project Proposal

Course: Machine Learning for Vision and Multimedia

Project: 2.6 — Few-Shot Logo Recognition

Group members:

Mehdi MEHADJI - s350601

1. Project Title and Goal

Title:

Few-Shot Logo Recognition with Pre-trained Visual Embeddings and Metric Learning

Goal:

We aim to build a few-shot logo detector/retrieval system that, given one or a few example images of a logo (the "query"), can retrieve all other instances of the same logo from a large image dataset, even when the logo appears under different scales, rotations, backgrounds or lighting conditions.

Concretely, our system will:

1. Learn a visual embedding space where images of the same logo are close, and different logos are far apart.

2. Use this embedding space to perform few-shot retrieval: starting from *K* support examples of a logo, retrieve all matching instances in a gallery set.

3. Optionally, localize the logo within images by using bounding box annotations or feature map similarity.

2. Dataset

We plan to use the **LogoDet-3K dataset**, as suggested in the project description.

- **Source:**

  LogoDet-3K: large-scale logo detection dataset with 3K logo categories and ~158K annotated logo instances.

  GitHub: https://github.com/Wangjing1551/LogoDet-3K-Dataset

- **What we will actually use (to keep the project manageable):**
  - We will select a subset of logo classes (e.g. 50–100 logos) with enough instances per class.
  - For each logo instance, we have:
    - The original image
    - The bounding box of the logo
    - The logo category label
- **Preprocessing & splits:**
  - Crop each annotated bounding box to obtain logo patches; optionally keep the full image for detection experiments.
  - Split classes into:
    - **Base classes:** many examples (used to train the embedding network).
    - **Novel classes:** unseen during base training, used only for few-shot evaluation.
  - Within each group (base / novel), we will create:
    - Training set
    - Validation set
    - Test set
  - The few-shot evaluation protocol will be episode-based:
    - For each episode, randomly sample *N* classes and *K* support examples per class (e.g. 5-way 5-shot).
    - Build a gallery of query/logo instances to retrieve from.

We will report dataset statistics in the final report (number of classes, instances per class, resolution distribution, etc.).

3. Model Architecture

We will follow the three phases requested in the project description.

Phase 1 — Baseline with Pre-trained Embeddings

Backbone:

- Start from an ImageNet pre-trained CNN, e.g. ResNet-18 or ResNet-50 (for speed, we prefer ResNet-18 at first).
- Input size: 224×224 RGB (standard ImageNet transforms).

Embedding head:

- Remove the original classification layer.
- Add:
  - Global Average Pooling (if not already present)
  - A fully-connected layer producing a D-dimensional embedding (e.g. D = 256 or 512)
  - L2-normalization of the embedding vector.

Usage:

- For each logo patch, compute its embedding.
- For retrieval, use cosine similarity / Euclidean distance between embeddings.
- For training the baseline, we will initially:
  - Freeze most of the backbone and train only a linear classification head on base classes (standard cross-entropy).
  - Then use the backbone as a frozen feature extractor for few-shot retrieval (Phase 2 baseline).

This directly uses transfer learning and feature extraction from the labs.

Phase 2 — Metric-Learning Few-Shot Model

To better support few-shot recognition, we will adopt a metric-learning strategy:

- Keep the same backbone (possibly partially fine-tuned).
- Replace the standard classifier with a metric-learning loss, e.g.:
    - Triplet loss (anchor–positive–negative)
    - Or Prototypical Networks style loss:
        - Compute class prototypes as the mean embedding of support examples.
        - Minimize negative log-likelihood of the correct class based on distances to prototypes.

This leverages concepts from non-sequential architectures and model surgery seen in the labs.

Phase 3 — Optional Localization / Detection

If time permits, we will extend from patch-level recognition to logo localization:

- Option A (lighter): Sliding-window / region-proposal on the feature map, followed by embedding similarity with the query logo.
- Option B (heavier): Fine-tune a pre-trained object detector (e.g. Faster R-CNN / RetinaNet with ResNet backbone) on base logo classes, and adapt it to novel logos using:
    - Class-agnostic detection head + embedding comparison, or
    - Few-shot fine-tuning of the detection head.

This step may be exploratory depending on time and computational resources.

4. Training Setup

We will implement everything in PyTorch, reusing patterns from the course labs:

Data preprocessing and augmentation

- Train transforms:
  - Resize → RandomResizedCrop(224)
  - RandomHorizontalFlip
  - Small RandomRotation / ColorJitter (if they don't alter logo identity)
  - ToTensor + Normalize (ImageNet mean/std)
- Validation/Test transforms:
  - Resize(256) → CenterCrop(224)
  - ToTensor + Normalize

    (no random augmentations to keep evaluation stable)

This applies on-the-fly data augmentation at each epoch, as practiced in the transfer learning lab.

Optimization

- Optimizer: AdamW or SGD with momentum
- Initial learning rates:
  - For warm-up / classifier-only training: LR ≈ 1e-3
  - For fine-tuning backbone: smaller LR (e.g. 1e-4 or 5e-5) to avoid destroying pre-trained features
- Learning rate schedule: StepLR or CosineAnnealingLR
- Batch size: chosen based on GPU memory (probably 32–128)
- Epochs: ~20–40 for classification baseline, fewer for episodic few-shot training (but more episodes).

Training variants / experiments

Required by the project and aligned with course labs:

1. Baseline feature extractor:
    - Backbone frozen, train only classifier head on base classes.
2. Fine-tuning whole network:
    - Start from baseline, unfreeze all layers, fine-tune with a small LR.
3. Fine-tuning last blocks only:
    - Unfreeze only the last residual block(s) + head → compare performance vs full fine-tuning.
4. With vs without data augmentation:
    - Repeat experiments with augmentations disabled to show overfitting and performance drop.
5. Few-shot evaluation:
    - Evaluate both:
        - "Naive" k-NN retrieval using baseline embeddings.
        - Metric-learning few-shot model (Prototypical / Triplet).

We will use train() / eval() modes correctly (BatchNorm, Dropout) as emphasized in Lab 4.

5. Evaluation Metrics

We will evaluate both classification and retrieval performance.

1. Classification metrics (on base classes):
    - Top-1 accuracy, Top-5 accuracy.
    - Confusion matrix to inspect errors.
2. Few-shot retrieval metrics (on novel classes):
    - Top-k Recall (e.g. Recall@1, Recall@5):

        - For each query, check whether the correct logo is present in the top k retrieved instances.
    - mAP (mean Average Precision):
        - Treat retrieval as ranking; compute AP per query and average.
    - Episode-based accuracy for Prototypical Networks (if used).
3. Optional detection metrics (if we complete localization):
    - mAP at IoU threshold (e.g. mAP@0.5) on logo bounding boxes.

For each experiment, we will:

- Plot training & validation loss/accuracy over epochs (as done in labs).
- Compare:
    - Frozen vs fine-tuned.
    - Partial vs full fine-tuning.
    - With vs without data augmentation.
    - Baseline vs metric-learning few-shot model.

6. Expected Outcomes and Risks

Expected outcomes

- A working prototype that:
    - Given K examples of a logo, retrieves other logo instances from a gallery.
    - Demonstrates that pre-trained ImageNet features + fine-tuning + metric learning provide good few-shot performance.
- Clear quantitative comparison of:
    - Different fine-tuning strategies.
    - Effect of data augmentation.
    - Baseline vs few-shot-oriented training.

Main risks / mitigations

- Dataset size & complexity: LogoDet-3K is large.

  → Mitigation: start from a smaller subset of classes and images; scale up if time allows.

- Training time: full fine-tuning of ResNet on a large dataset could be slow.

  → Mitigation: use ResNet-18, mixed precision training, and aggressive early stopping.

- Detection/localization complexity: object detection is more complex than patch retrieval.

  → Mitigation: treat detection/localization as an optional extension after patch-level retrieval works well.